Product Recommendations in E-commerce Systems using Content-based Clustering and Collaborative Filtering

Linda Hansson
atp10lh1@student.lu.se

September 7, 2015

Master's thesis work carried out at Apptus Technologies AB.

Supervisors: Erik Lindström, eriklmaths.lth.se
Edward Blurock, edward.blurock@mah.se

Examiner: Johan Lindström, johanlmaths.lth.se

Abstract

In this report we take a new approach to product recommendation. We investigate the the possibility of using a hybrid recommender consisting of content-based clustering and connections between clusters using collaborative filtering to make good product recommendations.

The algorithm is tested on real product and purchase data from two different companies - a big online book store and a smaller online clothing store. It is evaluated both for functionality as a backfiller to other algorithms and as a strong individual algorithm. The evaluation mainly looks at the number of purchases as metric but also uses accuracy and recall as evaluation metrics. The algorithm shows some promise for using it as an individual algorithm.

Keywords: Recommender systems, Content-based clustering, Collaborative filtering

Acknowledgements

I would like to thank the following people who have contributed to this work in one way or another:

Mikael Hammar at Apptus' for his advice, feedback and many good discussions

Edward Blurock and Bengt Nilsson from MAH for participation in the above discussions. I would also like to thank Edward Blurock for helping me become better at generalising

Björn Brodén at Apptus for help with understanding Apptus' system and helping with code for getting information from Apptus' system as well as making the baselines. I would also like to thank the rest of the people at Apptus for their help and making me feel welcome.

Erik Lindström at LTH for taking on the supervisor role for this thesis even though this is not his field of study and for providing stable support in the background.

I would also like to note that this research is part of the project "20130185 Förbättrade sök och rekommendationer för e-Handel " funded by a grant from the KK-foundation; see http://www.kks.se

Contents

1 Introduction **7**
- 1.1 Introduction to Recommender Systems . . . 7
- 1.2 Purpose . . . 8
- 1.3 Problem Definition . . . 8
  - 1.3.1 Aim of thesis . . . 8
  - 1.3.2 Limitations and Important Contributions . . . 9
- 1.4 Definitions . . . 9

2 Background **11**
- 2.1 Collaborative Filters . . . 11
  - 2.1.1 Common Methods . . . 11
  - 2.1.2 Groupings . . . 13
- 2.2 Content-based Filters . . . 13
- 2.3 Hybrid Filters . . . 14
- 2.4 K-Means++ Clustering . . . 15
  - 2.4.1 Advantages and Disadvantages . . . 16
- 2.5 Related Work . . . 16

3 Approach **19**
- 3.1 Clustering on specific attributes . . . 19
  - 3.1.1 The Apache Commons Math Machine Learning Library . . . 20
  - 3.1.2 The Overall Algorithm . . . 21
- 3.2 General Design . . . 24
  - 3.2.1 First Design . . . 24
  - 3.2.2 Working Design . . . 25

4 Evaluation **27**
- 4.1 Evaluation Metrics . . . 27
- 4.2 Data sets . . . 29
- 4.3 Apptus' Test Framework . . . 29

4.4 Baselines . . . 31
4.5 Test Methodology . . . 31
4.6 Results . . . 33
4.6.1 Interpreting Test Names . . . 33
4.6.2 Results from Attribute Specific Clustering . . . 33
4.6.3 Results for Company A . . . 34
4.6.4 Results for Company B . . . 36

5 Discussion **39**
5.1 Discussion of Results . . . 39
5.2 Future Work . . . 42

6 Conclusions **43**

Bibliography **45**

Appendix A The Rest of the Result Tables **51**
A.1 Results for Company A . . . 51
A.1.1 Individual Algorithm Results . . . 51
A.1.2 Results Using Synergy . . . 53
A.2 Results for Company B . . . 54
A.2.1 Individual Algorithm Results . . . 55
A.2.2 Results Using Synergy . . . 58

Appendix B How To Configure the Conditions File **59**
B.1 Formatting . . . 59
B.2 Transformation Classes . . . 60
B.2.1 TransformNValued . . . 60
B.2.2 TransformNValuedFilterPrefix . . . 60
B.2.3 TransformNValuedFilterContains . . . 60
B.2.4 TransformNValuedFilterNTopSeller . . . 60
B.2.5 TransformNValuedFilterNFrequent . . . 61
B.2.6 TransformNValuedFilterOutStopKeepNFrequent . . . 61
B.2.7 TransformNValuedFilterNFrequentTokenize . . . 61
B.2.8 TransformNumerical . . . 61
B.3 Filter Classes . . . 62
B.4 Clustering Classes . . . 62
B.5 A Detailed Configuration File Example . . . 62

Chapter 1
Introduction

Over the last years, ecommerce has been growing. In 2013, the total turnover for e-commerce in Europe increased with 17% compared to the year before[1] and big companies can have hundreds of thousands of products or more to choose from on their website. Both the customer and the company want the customer to easily find relevant products both during search and when simply browsing and this is where recommender systems comes into the picture.

1.1 Introduction to Recommender Systems

Recommender systems, in the most general sense, are used to give the user of a site/function (whether that be a streaming video site like Netflix and Youtube, an online shop like Amazon or an online auction site like eBay) some added value in the form of recommendations.

Some of the recommenders use only behavioural data, e.g. ratings, purchases, clicks and so on, to predict what items a customer would like. When doing predictions with little or no data about the users or items, these kind of recommenders usually can't give good, if any, recommendations. This is known as the cold-start problem [Meyer, 2012].

There are different kinds of systems and not everyone agrees on what classification should be used, but the most common classification divides all systems into one of three categories: Collaborative filters (CF), Content-based filters (CBF) or Hybrid filters [Meyer, 2012][Leander, 2014]. In [Meyer, 2012] a new classification is suggested that is based on the utility rather than the kind of information used by the system. The utilities are the following:

[1] Emota's "E-Commerce and Distance Selling in Europe Report 2014/15", pg.6

- Help to Decide
- Help to Explore
- Help to Compare
- Help to Discover

Recommender systems can also be categorised as giving non-personalised or personalised recommendations. Recommender systems providing personalised recommendations looks at a user's history to help make recommendations, while the type giving non-personalised recommendations does not take that into account.

Amazon is one e-commerce site that provides personal product recommendations. According to [Linden et al., 2003], they use item-item collaborative filtering in their recommendation system, i.e., they use behavioural data to make recommendations in the form of for example "people who bought this also bought that". [Nageswara and Talwar, 2008] mention that Amazon uses a hybrid recommendation system, i.e. a system that uses both content data and behavioural data. These will be discussed in section 2.3 Hybrid Filters.

1.2 Purpose

Most sites today use heuristic methods or algorithms using behavioural data in other ways in their recommender systems [Aldrich, 2015]. However, it could be of interest to explore other approaches where other methods, such as hybrid systems using both product data and behavioural data, are used to recommend products, as methods only using behavioural data usually have trouble recommending products when they don't have enough data [Meyer, 2012].

Also, much of the research done in the field of hybrid recommender systems, such as [Su and Khoshgoftaar, 2009][Kim et al., 2006][Tso and Schmidt-Thieme, 2006] have focused on ratings (either as the end goal or as a step to make recommendations) instead of direct top-*n* recommendations, which is what this project is focused on. The project is also using real data, which means problems will be encountered that would not be there on manufactured data, and has the advantage of using Apptus' test framework (this will be presented in the Evaluation chapter). Finally, one of the project goals is to be able to generalise the procedure created during the project enough to work on different sites without much configuration.

1.3 Problem Definition

1.3.1 Aim of thesis

The goal is to develop a hybrid model capable of making good product recommendations based on product information and behavioural data by clustering on that data, i.e., putting products together in groups based on how similar they are, and then recommending items

from the best clusters, given a product that belongs to a specific cluster.

The model should be general enough to work on a diverse number of sites, i.e. sites for e.g. clothing and books and sites with either big amounts of traffic or small amounts of traffic.

This leads to the following research question: Given a product described by a set of attribute-value pairs, is it possible that the use of a clustering algorithm (using the attribute-value pairs to compute distances) together with behavioural data (to find the best related clusters and best items to pick from those clusters), will lead to good product recommendations? By good recommendations we mean that the algorithm recommends the products that the customer eventually buys. In our case this will be tested in an offline setting, i.e., it will be tested on data generated from online stores but it will not be able to actually influence the customers as it would if the algorithm was tested online.

1.3.2 Limitations and Important Contributions

As the time for this project is limited, this work focuses only on non-personalised recommendations. Apptus, the company this thesis work was carried out at, has developed the testing framework that will be used to evaluate the algorithms. Björn Brodén at Apptus has also created some baselines that we will compare the results against.

While the work on the original algorithm, specific for one of the companies and testing clustering for two specific attributes, is my own work, the generalised version that extends this was designed jointly with the assistant supervisor for this thesis, Edward Blurock. He is also part of the bigger project this thesis is part of.

1.4 Definitions

Some of the vocabulary used in this report can be seen as ambiguous so this section is dedicated to explaining what the words will mean in this report.

An *attribute* has an *attribute name*, such as colour or title, and one or several *attribute values*, for example blue, “The Wind in the Willows”, etc. An *item* or a *product* is assumed to have several *attributes*, including an *identifier*. The identifier may also be referred to as a *key*, with or without the word *product* or *item* in front of it, in this report.

The *product catalogue* contains a set of *products*. Depending on the settings, it may be only *products* that have been bought before or the whole set. If nothing is said about it, it is assumed to be the whole set. *Items* and *products* will be used interchangeably even though *item* is a much broader concept.

eSales and *Apptus' test framework* will be used to refer to Apptus' system that contains all sorts of useful functions and classes. For example, if there is a line saying that *eSales* was used to get some information that means one or several of the functions in Apptus' system

was called to get such information.

The concept of a *wrapper* is used in this report. It is mainly used because *Apache Commons Math*^{TM} uses that in their example for their KMeans++ implementation. A *wrapper* is essentially an object holding together related information in a packaged format, very much like a class in Java.

When talking about *clusters*, the word *bucket* will sometimes be used. A *bucket* is basically a *wrapper* for a *cluster* but sometimes it contains a little extra information. For this report however, they can be treated as the same thing.

In the second step of the algorithm, we will use behavioural data to find *connections*, or *associations*, between items. Those words will be used interchangeably in this report. An item is *connected* to another item if they have been bought by the same person in the same session. The more times they have been bought together, the stronger the *connection* is.

Chapter 2
Background

This chapter will introduce the reader to some of the more theoretical background, including the different kinds of systems, related work and the clustering techniques we will be using in this project.

2.1 Collaborative Filters

Collaborative filters (CF) use information given by the user. It can be either implicit information in form of clicks, add to carts or purchases for example, or explicit information such as ratings of an item [Rendle et al., 2009]. This data is also called behavioural data. As written before in the introduction, this kind of filter often suffers from the cold-start problem, but it usually works well if it has enough behavioural data [Meyer, 2012].

2.1.1 Common Methods

The two most popular methods for collaborative filtering are k-Nearest Neighbour (kNN) and Matrix Factorisation (MF) [Rendle et al., 2009].

K-Nearest-Neighbour finds the k nearest neighbours to an item or a user and uses information from the neighbours in some useful way, it can for example use the ratings of the neighbouring items to predict the ratings of another item [Töscher et al., 2008].

In the simplest application, there are two classes an item can belong to and the k-nearest neighbours (who are already classified) will decide which class a new item belongs to. The item then belongs to the class most of its neighbours belong to; see example in figure 2.1 where k is set to 3.

Matrix Factorisation is mainly used for predicting ratings of items and uses a matrix to

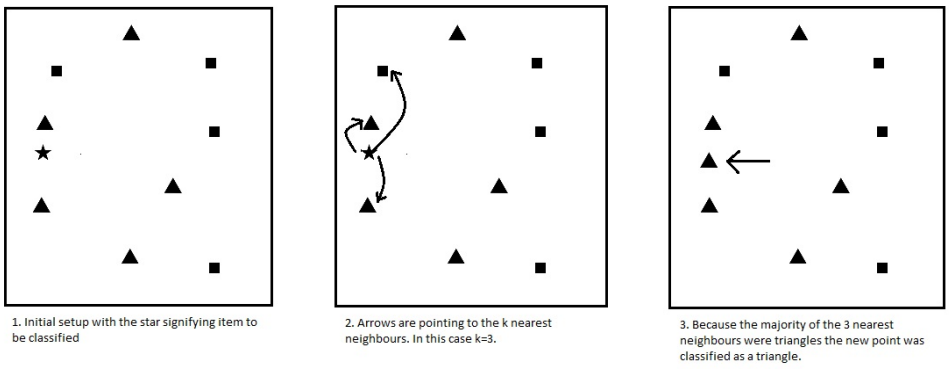

Figure 2.1: Example of knn with 2 classes and k set to 2.

represent the ratings of items by users. The problem can be seen as fitting a linear model to the user-item matrix Y so that $Y \approx UV$ where U and V are matrices that need to be found [Wu, 2007]; see figure 2.2.

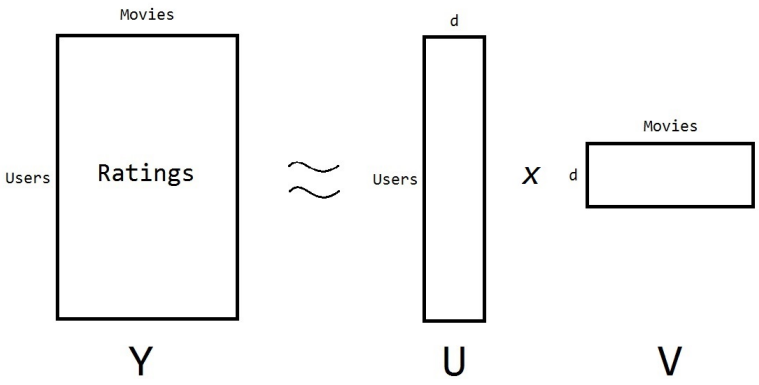

Figure 2.2: Figure of matrix factorisation using movie ratings where d is the number of latent factors used in the factorisation. The numbers in the U matrix indicates the users' interest in the latent factors. The numbers in the V matrix indicates what kind of relationship there is between the movies and the latent factors.

Matrix Factorisation is a latent factor model. For the user-item rating case this means it uses the ratings to find latent factors for the users and items and builds the vectors of matrices U and V on that. Latent factors can both be in a dimension that is easily understandable and more abstract. Easily interpretative latent factors could for example be serious movies vs comedy ones, a measure of attributes such as how many action scenes or how much and what kind of music is in the film or independent films vs. large-budget films [Bell et al., 2010].

Matrix Factorisation is not adapted to item-item recommendations but in the case of rat-

ings it is usually slightly better than kNN [Meyer, 2012]. kNN methods used by themselves have problems with scalability as their computational time is $O(n^2)$, where n is the number of items or users to be clustered. There are however ways to improve the scalability, e.g. by using another clustering technique, such as kMeans, to do the clustering part of the algorithm offline [Meyer, 2012].

2.1.2 Groupings

Collaborative filters are usually divided into two sub groups: memory-based and model-based filters. The memory-based ones use the data directly while the model-based ones use the data to build a model that can make predictions by exploiting the user-item interactions [Jiang et al., 2013]. Model-based techniques include clustering models, Bayesian belief nets and SVD-based models (including MF) [Su and Khoshgoftaar, 2009][Jiang et al., 2013]. Memory-based techniques include the neighbourhood-based collaborative filtering algorithm [Su and Khoshgoftaar, 2009].

Collaborative filtering can also be divided into user-based and item-based filters [Tso and Schmidt-Thieme, 2006]. The user-based techniques generally looks at similarities between users, e.g. by computing similarity measures based on how the users have rated different items [Rendle et al., 2009]. The item-based techniques looks at similarities between items [Sarwar et al., 2001].

For kNN-based techniques, similarities computed based on item-item usually give better results than the user-user ones [Rendle et al., 2009] [Meyer, 2012][Hu et al., 2008].

2.2 Content-based Filters

Content-based filters use information about the items/products to generate recommendations. Some of the techniques used include k-Nearest Neighbours [Lops et al., 2009], Bayesian classifiers, decision trees [Adomavicius and Tuzhilin, 2005] and the Vector Space Model [Lops et al., 2009]. CBF emerged from the fields of information retrieval and information filtering research [Adomavicius and Tuzhilin, 2005].

According to [Sarwar et al., 2001], content-based methods can increase the coverage of a recommender, i.e. increase the number of different products that are recommended and cover more of the product catalogue. However, since the similarity is based on the items' similarity to the reference item, recommenders based on content based filters tend to recommend items that are very similar to the reference one [Meyer, 2012] and miss connections such as if you are buying a mouse, you might want to buy a mouse pad or a keyboard as well. The tendency to recommend very similar items is also called overspecialisation [Meyer, 2012].

Given that Content-based Filters works on similarity of items, there is an assumption that customers would either be interested in consuming similar items to those they have already consumed or placed in their cart (upselling) or are interested in alternative similar items to those they are currently considering (alternative recommendations).

Unlike Collaborative Filters, Content-based filters do not suffer from the cold start problem caused by lack of behavioural data since it only uses content-based data. However, it is very much dependent on the quality and structure of the content data since that is all it uses. If the content data has many missing or corrupted values, this will affect the filter.

2.3 Hybrid Filters

What constitutes a hybrid filter and how they should be classified is not entirely clear cut. Some, for example [Nageswara and Talwar, 2008] and [Kim et al., 2006], specify a hybrid filter as one that uses both collaborative filtering and content-based filtering. Others, such as [Su and Khoshgoftaar, 2009], also allow for combinations of collaborative filters to be specified as hybrids. [Meyer, 2012] specifies a hybrid system to be ”a system based on different data sources of different natures”.

[Burke, 2002] suggests a classification of seven types of hybrids:

- Weighted – scores from different models are combined to produce recommendation
- Switching – uses different models based on some criteria
- Mixed – recommendations from several models are used
- Feature combination – e.g. using collaborative features of items in a content-based model, or raw data from models are used together in one algorithm
- Cascade – one recommender refines the result of another
- Feature augmentation – one model takes part of its input from the output (could be several generated features) of another model
- Meta-level - the model learned by one recommender is used as input to another

Meyer has reviewed Burke’s classification and building on that instead proposes a classification with four types (or families as he calls them), see [Meyer, 2012].

By using hybrid filters one can decrease the impact of each of the individual filters shortcomings [Adomavicius and Tuzhilin, 2005], e.g. by combining a collaborative filter and a content-based filter it would help with the cold-start problem the CF suffers from and the overspecialisation the CBF suffers from. Several papers have also shown that the performances of hybrid filters are better than that of their individual counterparts [Adomavicius and Tuzhilin, 2005].

2.4 K-Means++ Clustering

In this section we will first talk about k-Means, the algorithm k-Means++ evolved from, and then talk about how k-Means and k-Means++ differs and the advantages and disadvantages of k-Means based algorithms.

K-Means is a non-hierarchical clustering method that given n points and an integer k will group the n points into k clusters [Huang, 1998]. It can only handle numerical values (or categorical values transformed into numerical ones) [Huang, 1998] and works by minimising the average squared distance between points in the same cluster [Arthur and Vassilvitskii, 2007], which can also be seen as minimising the within groups sum of squared errors (WGSS) [Huang, 1998]. We will from now on mostly use WGSS when referring to it.

The algorithm has the following steps:

1. Assign initial centers, usually chosen uniformly at random from the points.
2. For each point, calculate distance to each center and assign it to the nearest one.
3. Recalculate the centers' positions so they are the center of mass of all the points they have been assigned

Repeat steps 2 and 3 until the positions of the centers no longer change. [Arthur and Vassilvitskii, 2007] Figure 2.3 illustrates the algorithm for $k = 3$.

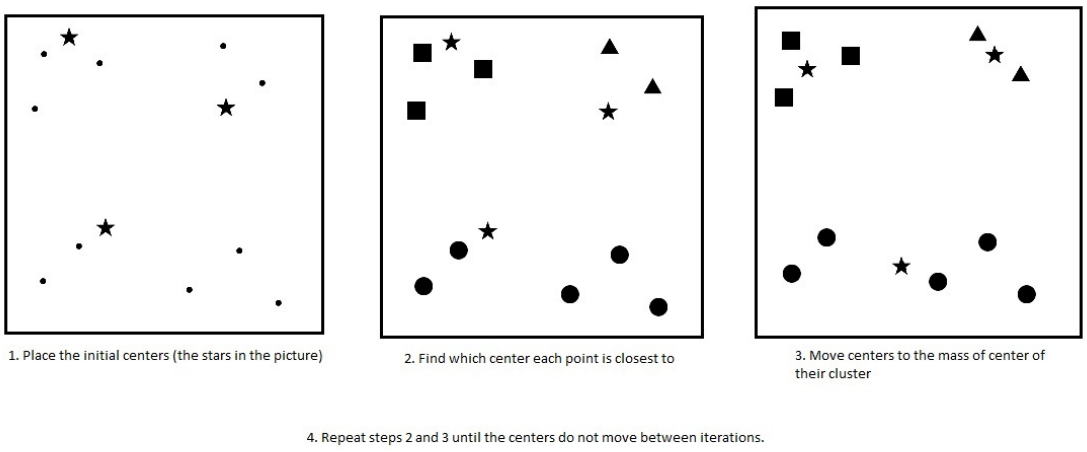

Figure 2.3: Illustration of the k-Means algorithm with k set as 3.

The algorithm's computational complexity is $O(Tkn)$ where T is the number of iterations (steps 2 and 3), k is the number of clusters and n is the number of points to cluster [Huang, 1998].It has been proven that for each iteration WGSS is either decreased or the same [Arthur and Vassilvitskii, 2007].

[Arthur and Vassilvitskii, 2007] came up with an algorithm they decided to call k-Means++. k-Means++ is like k-Means except it uses another technique to place the initial centers. The first point is still chosen uniformly at random, but for the other k-1 centers, the following defines the selection:

Let X be the set of points to cluster and $D(x)$ be the distance from point x to the center it is closest to of those centers already chosen. Then, each point $x \epsilon X$ has a probability $\frac{D(x)^2}{\sum_{x \in X} D(x)^2}$ of being chosen each round a center is selected.[Arthur and Vassilvitskii, 2007]

In [Arthur and Vassilvitskii, 2007], it is proved that k-Means++ is $O(\log k)$ competitive, i.e. that the WGSS is at most $O(\log k)$ times the WGSS of the optimal solution. They also show that k-Means++ is both significantly faster and the WGSS significantly smaller than for k-Means, on synthetic data sets as well as on real ones.

2.4.1 Advantages and Disadvantages

The advantages of the k-Means algorithm are that it is fast, simple [Arthur and Vassilvitskii, 2007] and can process large data sets efficiently [Huang, 1998].

The disadvantages are that the minimum it finds is only a local minimum [Huang, 1998] and it can generate bad clusterings, i.e. its accuracy is not that good compared to other algorithms for creating clusters [Arthur and Vassilvitskii, 2007]. Another problem is that the algorithm needs the number of clusters it should divide the points into, it can't compute the best possible value for k by itself.

2.5 Related Work

In my search for a hybrid system using implicit feedback (in collaborative filtering) and content-based filtering for recommending top-N items, I have not been able to find many systems. Therefore, I will go through both the few I found as well as a few other systems that are related to just part of my work.

[Gantner et al., 2010] implemented a hybrid using implicit feedback and content-data where the content data was used to alleviate the cold-start problem. The implicit feedback was used in their matrix factorisation based on Bayesian Personalised Ranking and then the content data was mapped to latent factors for the items/users that didn't have implicit feedback data. The existing latent factors was used to find the mappings needed to compute the factors from the content data.

SSLIM [Ning and Karypis, 2012], a "set of sparse linear methods with side information" utilizes purchase information and item information to get top-N items. This is an extension of their earlier work on a method called SLIM. SLIM uses a customer's previous purchase history to get recommendation scores for all items that have not been purchased by that customer, using an estimated aggregation coefficient matrix. The new set of methods uses

the side information in different ways when estimating the aggregation coefficient matrix and showed improvements compared to SLIM.

SPrank (Semantic Path-based ranking) was the first hybrid system to use implicit feedback and content-based filtering utilizing the Web of Data to recommend top-N items [Ostuni et al., 2013]. In SPrank, both the Linked Open Data and the data model for the CF algorithm are seen as graphs and they are merged to create one hybrid algorithm utilizing both the links between content data and the implicit feedback [Ostuni et al., 2013].

[Peska and Vojtas, 2014] did offline experiments on a travel agency data set. They implemented Forbes & Zu's content boosted factorization method and used implicit feedback (among others visited pages) and attribute information as input to the matrices.

[Hu et al., 2008] looked at using implicit feedback in item-item kNN and SVD to recommend TV shows, but did not use any content-based information, thus making it a pure CF recommender system.

In [Leander, 2014], implicit feedback in form of clicks and purchases are used to find links between product properties in an online clothing store. The work focused on getting both high accuracy and high coverage and the results indicated that it was possible to get good coverage without affecting the accuracy too much.

In Fab, a hybrid web page recommender system described by [Balabanović and Shoham, 1997], content-based filtering is used to collect web pages that are related to one or several topics chosen by users. Collaborative filtering is then used on the individual user level to filter out pages already seen and limit the number of pages from the same site in the same recommendation batch, but also to forward pages rated highly by similar users.

[Oard and Kim, 1998] were one of the first to write about a general approach for how implicit feedback could be used for recommendations and built on Nichols work on implicit sources. Others before them who looked at implicit feedback in recommender systems using specific sources include Karlgren, Morita & Shinoda, Konstan et al and Rucker & Polanco. [Oard and Kim, 1998].

Later contributions to the field using implicit feedback include [Rendle et al., 2009] where they use Bayesian Personalized Ranking, using clicks, to determine the ranking of items. Rendle and Freudenthaler continued this work and in [Rendle and Freudenthaler, 2014] proposed using non-uniform samplers to speed up the convergence of the Bayesian Personalized Ranker. Something to keep in mind, however, is that there is a big difference between using clicks and payments to determine rankings as payments are expensive for a user while clicks are not. One could make an analogy to how players' behaviour changes when playing poker with real money instead of fake.

Chapter 3
Approach

In this chapter we will present to you the approach we took to implement the hybrid system. In the first section, we will describe how we made a solution specific to one of Apptus' customers using only two specific attributes to do the clustering. Next we will describe how we decided to design our general system and the bumps in the road on the way there.

3.1 Clustering on specific attributes

To start with something, we decided to do a clustering on two specific attributes, one categorical and one numerical one. The attributes will in this report be called Attribute A and Attribute E respectively. At first we were going to use kNN but then remembered that to use kNN you need to already have classifications for the points used in the training set and we did not have that. Then we looked at a couple of different options including k-Means, DBSCAN and hierarchical clustering.

There are two different types of hierarchical clustering (agglomerative and divisive). The type determines in which order the clusters are made but what they have in common is that they use a distance function to determine how to form the next cluster and the end result is a hierarchy of clusters of different sizes. Hierarchical clustering was ruled out because we wanted to be able to use different attributes on different levels and not just combine them in one metric.

DBSCAN creates clusters by picking points randomly and then sees if their near neighbourhood (specified by user) has enough points (also specified by user) to create a cluster from that. If so, the neighbourhood's near neighbourhoods are also included in that cluster if the point's neighbourhood has enough points. Otherwise, the point is temporarily labelled as an outlier (or noise) and another point is picked. The temporary outliers that have not been put in a cluster when all points have been visited becomes permanent outliers.

DBSCAN would have been a good alternative if it weren't for the fact that it leaves outliers and we wanted the algorithm to cluster all the points we sent into it. One big advantage to using DBSCAN would have been that we wouldn't have to specify the number of clusters.

In the end our choice fell on k-Means (or rather k-Means++, a more efficient version of it), partly because it is well known and partly because it would be easy to explain the workings of it to Apptus and their customers. Another relevant point for using k-Means was that other people in the research project wanted to do fuzzy clustering at a later point in their project. The advantages and disadvantages of the algorithm are explained in 2.4.

We looked at a couple of different options for working with the algorithm, among those R packages that implement k-Means, e.g. flexclust, but decided against using R - primarily because hardly anyone in the research project besides me had worked in R before. In the end we decided to use Apache Commons Math Machine Learning Library since it uses Java and that was the common language shared by the members of the research project and the language most of Apptus' system is written in.

We will spare you the trials and tribulations we went through to parse the big product catalogue (at this point we weren't working directly in Apptus' system yet) and build up an item representation that was much more memory efficient than our first attempts. Suffice it to say that once we were done parsing and storing the items we had a much better knowledge of how to read and write xml files and how a generic memory efficient system for storing items could be done. The system built on having a central object that contained two ArrayList objects, one for the attribute values and one for the attribute names and then each item object referred to their values by storing the integer that corresponded to the value's index in the relevant list.

3.1.1 The Apache Commons Math Machine Learning Library

We decided to use the Apache Commons Math Machine Learning Library because it had the k-Means, or rather k-Means++, algorithm already implemented. To use their implementation of the algorithm, each point has to be an instance of a class that implements their interface Clusterable. This basically means that a wrapper class has to be implemented where an item is represented as a vector of double values. Then each item that you want to be clustered has to be transformed into such a wrapper. The algorithm, called by creating a KMeansPlusPlusClusterer object and then calling cluster on it with a list of wrappers as input argument, returns a list of CentroidCluster objects that contains information about the centroid and the wrappers that belong to the centroids cluster.

The Apache Commons Math Machine Learning Library also has implemented algorithms for doing fuzzy k-Means clustering, DBSCAN and Multi-k-Means++ clustering. Multi-k-Means++ clustering is basically like k-Means++ clustering but it does the clustering several times and has an evaluator that decides which of the clusterings was the best and returns that.

3.1.2 The Overall Algorithm

The overall algorithm consists of two parts, the clustering part and the recommendation part, and it is also used in the general implementation. Before the clustering, the items that we are going to cluster needs to be preprocessed. During the preprocessing, wrapper objects (of the wrapper class that implements Apache's Clusterable interface) are created. Since k-Means++ and thus Apache's implementation of it only works with numerical values we had to find a way to represent our categorical values in a double vector. This was done by letting each double element in the vector represent a boolean value signifying whether the item had a specific value for an attribute or not. If we would have had the attribute colour with four possible values – red, blue, yellow and black, and one item had the values yellow and black, we would get a double vector with the elements 0,0,1,1.

The clustering itself is also done in two steps. First clustering is done on categorical Attribute A. This entails the following:

1. All items who have more than one value of a certain type for this attribute is sorted out and clustered by Apache's algorithm.

2. The product that weren't part of the clustering but have one value of the right type equal to any of the items that were clustered are placed in a cluster by classifying them (more on classifying items later)

3. The products that did not have any common values of the right type with the ones clustered are gone through separately and clusters are created separately. For each product it checks if a cluster has been created with same value of the right type, if so it puts the product in that cluster. Else it creates a new cluster, puts the product in there and adds the cluster to the separate clusters.

4. The two cluster groups are put together into one group.

In the second step the clusters that have more members than a predetermined threshold, in this case 50, are selected for a second round of clustering using numerical Attribute E. Those clusters are removed from the group of clusters and replaced with the clusters generated during this second clustering on the numerical attribute. We chose to set the threshold at 50 because it seemed a good size to stop at. If the threshold had been smaller we would have run the risk of making the clusters too small for recommending items from within them and setting the threshold too high would leave the selection too great.

The clustering on the numerical attribute does not use k-Means++ but instead, for each cluster, creates a definition of buckets (low and high limits) and then places each item in the cluster into a bucket based on their value for the numerical attribute. The items that don't have a value for the attribute end up in a separate cluster with "NO_PRICE" appended to its id. Attribute E only has one value so no handling of several values is needed.

The reason why only values of a certain type were chosen is because when we did some clustering at the beginning of the project with this attribute we found that we got less noise doing that, the algorithm found more clusters that made sense instead of many clusters with

centroid that had small frequencies of many attribute values.

Some thought was put into how k should be chosen for the k-Means++ algorithm. In the early stages our plan was to clustering on several different k's, calculate the within groups sum of squares and then plot that to determine which k to use or calculate it from the within groups sum of squares without having to plot it. We implemented the calculation of the within groups sum of squares but it slowed down the program a lot (it more than the doubled the time) so we decided not to use that. Instead we fell back on a simple rule of thumb: $k = \sqrt{\frac{n}{2}}$, where n is the number of items to cluster [Mardia et al., 1979].

Two classifiers were built to be able to classify items - one for just clustering on Attribute A and one extended to handle clustering on both Attribute A and Attribute E. What they both do first is use the distance measure of Apache's algorithm to calculate distances from the item to each cluster and then sort those distances together with the cluster ids in a list in increasing order. Then the classifiers check if there is a tie for smallest distance. It is how the classifiers handle ties that sets them apart.

The one-level classifier (handling only clustering on Attribute A) picks one of the tied clusters at random and returns the id of that cluster. The other classifier begins by checking if the item has a value for that Attribute E. If it doesn't, it checks if there are any tied clusters with "NO_PRICE"-appended to their id and in that case chooses that one, otherwise it chooses one at random.

If the item does have a value for the attribute it finds its bucket index, then goes through all the tied clusters and checks if they are sub-clustered (done by checking if they were an instance of a subclass to the otherwise used cluster class). If they aren't, the process stops and one of the tied clusters are chosen at random. If they are, the bucket index difference between the item and the cluster is calculated and the id of the clusters with the lowest difference are kept. Once every cluster has been gone through, if there was only one lowest difference, it returns the corresponding cluster id. Otherwise it chooses one of the ids with lowest difference at random.

The classifiers were tested by classifying all the items to see how often they gave back the correct cluster id. The one-level classifier had a misclassification rate of approximately 33%. A closer inspection showed that the items that were misclassified all belonged to the same cluster and that this cluster was a really big one with a centroid that had many small frequencies. They were misclassified into clusters that had one or two value frequencies in their center or no value frequencies. As we thought this might actually be a better classification for them than the big cluster we decided to keep it this way. The other classifier had a misclassification rate of approximately 28%. Considering that it had the same sort of misclassification errors as the first classifier we were happy with the level of misclassifications.

Each level of clustering is in most cases done only once, but for some attributes in the general implementation, where purchase data is used in the clustering, it is done more often. The recommendation part of the algorithm is called every time a recommendation is requested based on a viewed item. First the item viewed will be classified into one of the

clusters and then that information will be used in what I have chosen to call a connection algorithm. We have implemented three different connection algorithms (described below) that will use purchase data to find connections and from that recommend items.

The reason why we are doing these two parts is that the clustering is content-based and if we were to only recommend from within the cluster we would get the problem of over-specialisation and miss highly connected items that are not similar in content. This is why we have the connection algorithms. The reason why we are doing the clustering is to be able to give recommendations to items that have not been bought together with anything or with very few items. This is where a classic collaborative filter would come up short.

Connection Algorithm 1

The first connection algorithm does not really live up to its name as it just takes the cluster an item was classified into and recommends the top-*n* from the same cluster. It is a seemingly simple algorithm that does not require any calculations of sales figures between clusters. However, in the implementations done by Apptus to get top-*n* of different kinds (e.g. top seller, revenue-based, etc) complex factors such as aging is used to give a better result than one would get if simply giving each sale the same importance. As mentioned, it can use several implementations, among those top-*n* based on revenue, sales, etc. For our implementation, it uses the number of sales of each item to determine top-*n*. This algorithm might later in the report also be referred to as the within-cluster algorithm.

Connection Algorithm 2

The second connection algorithm is the most complicated. As the first one, it takes the cluster an item was classified into as the input. It then looks at each item in the cluster and calculates their connections to other items in other clusters and adds those figures together. In the end, this gives us the items in the product catalogue that are overall most connected to the items of the cluster and these are what is used as the recommended products. Just as in algorithm 1 our implementation uses number of sales to decide how strong a connection is. This algorithm also allows connections between items in the same cluster so the items in the cluster the original item was classified into are not ruled out as candidates. This algorithm might later also be referred to as the cluster-to-item algorithm.

Connection Algorithm 3

The third algorithm is similar to the second one but instead of looking at cluster-to-item we are looking at cluster-to-cluster. Just as before, all items in the cluster an item was classified in is gone through, but now we are looking at how strong their connection is to each cluster, i.e., how many products in each cluster it is connected to and how strongly. In our implementation, the strength of the connection is again decided by how many times the items were bought together. The figures from each item is then added up and top-*n* computed among the items in the cluster it has the strongest overall connection to. Top-*n* is in our implementation based on number of sales. This algorithm might later be referred to as the cluster-to-cluster-algorithm.

A variant of this algorithm where several strongly connected clusters were looked at and the strength of the connections and sales figures decided top-n was also considered but not implemented in this project because of time constraints.

3.2 General Design

In this section we will talk about how the general implementation is designed and the choices made along the way. Appendix B goes into more detail of the classes available to use in the configuration and how to write the configurations file for the system.

As mentioned earlier, the overall algorithm with clustering and then finding connections is the same in the general design. To make the algorithm more general however, the way we did the clustering needed to change. To continue using the implementation of kMeans++ we had used so far we had to come up with a better solution for storing an item's values in a double vector. Unlike before when we only had two attributes, we were now dealing with an unknown number of attributes with an unknown number of possible values for them. This meant we couldn't just create one long double vector with some variable to keep track of which part of the vector belonged to which attribute, because that would have quickly caused the system to run out of memory.

3.2.1 First Design

In the first attempt at a general design, our goal was to make pretty much the whole process automatic. Weights for each of the attributes would be calculated by running the algorithm on different settings, hopefully using some clever algorithm to minimise the number of weight configurations needed, and then using the weight vector that gave the best result on recommendations.

However, we still had to solve the problem of how to store values from several attributes in one double vector without taking up too much space. The plan was to keep the way that the item information was stored in the item object (referencing attribute names values by indices to save space) and simply change how the wrappers were created.

Our idea was to let each vector element represent one attribute and encode the values in the double value stored there. Apache's algorithm allows making own implementations of the class that calculates the distance provided that the method takes two double vectors as input and returns a double value. This meant we could send in two double vectors that had encoded values, decode them and then use e.g. $d = 1 - J$, where J is the Jaccard similarity coefficient, as distance measure and incorporate the weights into the measure. In mathematical notation the Jaccard similarity coefficient can be described as $J = \frac{|A \cap B|}{|A \cup B|}$, where A and B are sets of values.

The encoding was implemented using the indices that referenced the actual values and padding each encoded attribute value with zeros at the beginning so all the values had the

same length and then they were concatenated into a string and transformed back into a double. The double value also contained information about how many digits were used to represent one value. We discovered pretty quickly when we ran the code that something was wrong and found that not only had we forgotten to take into account that the algorithm would make gibberish out of the centres' encoding but we could not use the encoding at all since it required too many digits to work as a double value.

3.2.2 Working Design

Going back to the drawing board, there were two solutions we had in mind. The first one meant abandoning Apache's algorithm and implementing our own version of k-Means++, or rather something like kModes that is more suited to categorical data. The second one meant sticking with Apache's algorithm and building up a framework where the user specified which attributes to use and (choosing from a number of transforms) how the values of an attribute should be transformed into a double vector. For the categorical values this meant putting restrictions on how many different values to represent if there were many values overall to make it possible to run the algorithm.

Since we were undecided and our assistant supervisor Edward Blurock, who is also part of Apptus' research group, wanted us to go with the second alternative, we ended up choosing that one. Because the design of the second option was mainly our assistant supervisor's, the implementation of this design was done together with him. Although we have added on quite a few transform classes after the initial system implementation was done, most of the system implementation was done together with our assistant supervisor and done in such a way that there aren't any parts of it we can take sole credit for.

In this section I will describe the design of the system and its advantages and limitations. The system is quite complex as we decided to add on quite a bit of functionality and generalise it to a much greater extent than we would have previously expected to do.

Overview of Design

The original design from the clustering on specific attributes was largely kept but became a much smaller part of the overall solution. In the general solution, we decided to make it possible to do more levels of clustering, i.e. more than the two used before and also have the choice of how the clustering should be made and which variables to use.

Each level of clustering is basically its own unit that sends back its results to a class that keeps track of all the levels of clustering. This class is called ListofLevelHierarchies. Getting and transforming the item data into a suitable format for the clustering is done at each level. The reason for this is that different attributes (as well as ways to transform the data) and cluster techniques might be used at different levels.

ListOfLevelHierarchies keeps track of the levels, initialises them and handles the clustering on several levels by sending on clusters to be sub-clustered to the right levels and keeping all the cluster results in a tree structure. It is also the class that handles the top

level part of classifying an item into a cluster. The actual classification is done by the classes contained in the tree structure holding the cluster results and is done in such a way that it allows for fuzzy classification (used with fuzzy clustering) with an easy extension.

The Transform classes take care of transforming the item data into something a cluster algorithm can use. Much of the code for transforming values of the same type as Attribute A into data that could be used by Apache's KMeans++ algorithm was kept and so was the code for bucketing items with numerical data as Attribute E. Also the code surrounding the k-Means++ algorithm, i.e. only use those wrappers that have several values in the initial clustering etc., was kept except in a more generalised form where it looked at what conditions the user had specified for that level and attribute.

All the levels, attributes, transforms (to use to convert the item data into a clusterable format) and conditions on the items are specified by the user in something we have chosen to call a conditions file. At the start of the system, this conditions file will be processed and the transforms initialised. Information about the different transforms and clustering techniques implemented and how to configure the conditions file can be found in Appendix B.

The choice of k for the k-Means++ algorithm was mostly kept the same way as before, i.e. $k = \sqrt{\frac{n}{2}}$. However, for the second data set, where some of the categorical attributes were found to have very few values compared to the number of items, we had to change the choice of k slightly to avoid getting empty clusters. First of all, after each cluster level is done, the system will check if any of the resulting clusters are empty ones and remove those. Secondly, there is a mechanism in place when k is determined that checks the size of k against the number of possible center choices. To simplify things, it assumes only one value per attribute can be chosen. If k is bigger than the number of possible center choices, k is set to that number instead. This also serves to speed up the clustering process since the algorithm will spend less time computing the initial centres if there are fewer to compute.

Advantages

The advantages of this system are that it is very general and it is possible to work on many different data sets and choose whatever attributes you want as long as Apptus' frame work can filter by it. In addition, any number of layers is possible as well as putting restrictions on content and number of values an item should have for an attribute. It is also easily extended to include new transforms and clustering techniques.

Limitations

One big limitation of this general system is that one needs to choose attributes and transforms for it manually. Another is that the clustering can take time because of the amount of vector elements used. Also, since the way the values are transformed require a lot of space, restrictions such as only use n most frequent values have to be enforced which leads to loss of data.

Chapter 4
Evaluation

In this chapter we will describe how we evaluated our algorithms. The first section will be about which evaluation metrics we have used and why. Then we will describe the data sets we have used, how Apptus' Test Framework works, our choice of baselines, test methodology and finally, the results.

4.1 Evaluation Metrics

The metrics we have looked closest at during our testing have been *purchases*, *precision* and *recall*. Purchases shows how many items of those we recommended were actually bought. Recommendations of an item after that item has been clicked will not count as a purchase based on recommendation. The purchase metric also shows a breakdown of the purchases into which position they were recommended at. We chose to use purchases as one of our primary metrics in this evaluation because in the real world it is important that a new solution increases the sales and revenue compared to the old solution.

We found figure 4.1 [1] to be very good for understanding *precision* and *recall*; see footnote for attribution of figure. *Precision* can also be described as the part of the recommended items that were actually bought while *recall* can be described as the part of all purchases that were recommended. We chose to use these two metrics because they are used by Apptus in their test framework and also commonly used when evaluating how good an algorithm is at recommending items [Leander, 2014].

However, since we are dealing with a ranking problem and not a classification problem, Apptus has modified their versions of precision and recall. They have redefined recall and

[1] Attribution: By Walber (Own work) [CC BY-SA 4.0 (http://creativecommons.org/licenses/by-sa/4.0)], via Wikimedia Commons

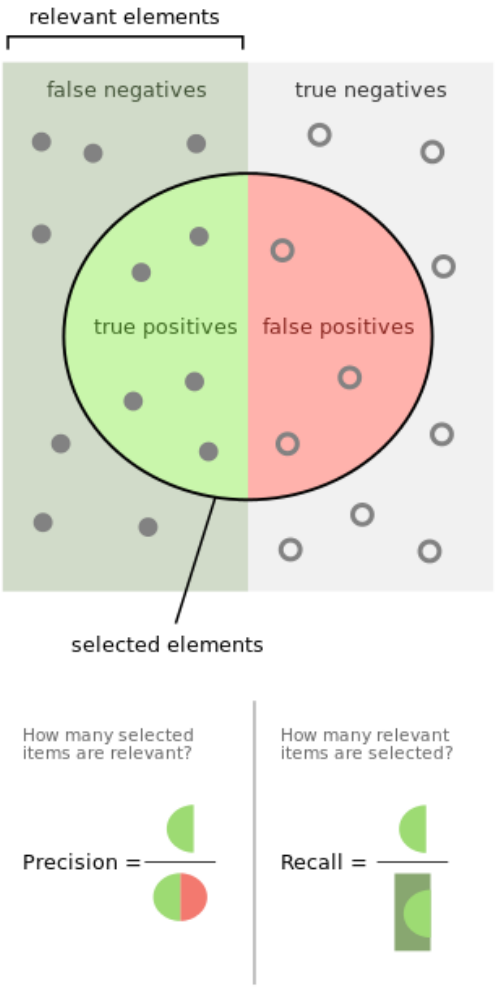

Figure 4.1: Figure explaining concept of precision and recall. By Walber (Own work) [CC BY-SA 4.0 (http://creativecommons.org/licenses/by-sa/4.0)], via Wikimedia Commons. In our case the true positives are the instances where one of the items that were recommended were actually bought while the false positives are the times products were recommended but were not bought. The false negatives are the products that were bought but were not recommended and the true negatives are the products that were not bought and were not recommended. When going from offline to online testing, which recommendations produce false negatives and false positives could very well change since the user would be affected by the algorithms recommendations. False positives are also called type I errors and false negatives are called type II errors.

precision as follows:

$$precision = \frac{\#hits}{\#displays}$$

$$recall = \frac{\#hits}{\#queries}$$

Precision is a measure of how good a specific product recommendation is where $\#hits$ is the number of times an answer to a query led to a purchase while $\#displays$ is the number of times we have answered a query. This means that an unanswered query does not impact the precision score negatively.

Recall measures how many of the relevant queries that had a hit. Here a hit means that something that was recommended in response to a query was bought later in the session and a relevant query is any query in a session where a payment was made. This means that an unanswered query will impact the recall score negatively.

4.2 Data sets

The algorithm was tested on two different data sets belonging to two of Apptus' customers. Because of confidentiality we will refer to them as Company A and Company B. Company A is an online retailer of books while Company B is a fashion retailer. The data sets are comprised of a product catalogue and event files. Company A had a very large product catalogue, too big to simply use straight off, so we picked out the subset of products that had been bought during the time frame the event data came from. Company B's dataset was much smaller. We kept it as it was because then it was about the same size as Company A's reduced one.

There is no noticeable difference in sparseness for the event data, i.e. they both have the same amount of purchases per number of event packets. One event packet is roughly one event and an event could be a click, a display, a query, an add to cart or a payment. Once we were testing, the test framework was configured to only include click and payment events.

Only sessions that led to a payment were included in the final data sets.

4.3 Apptus' Test Framework

According to [Cremonesi et al., 2008], there are three traditional data partitioning methods that are used when evaluating algorithms - holdout, leave-one-out and m-fold cross-validation. Holdout partitions the data set into two parts, train and test set, where the ratios used to divide the set vary from article to article[Cremonesi et al., 2008]. Cremonesi's description of leave-one-out seems to assume we have several points of information per user (typical for data sets with ratings). It chooses a user, takes out one of their data points and then tries to make a prediction for that user using all the remaining points from both that specific user and all the others. This is done for all the users and then the final performance

is the average of the performance of all these predictions. As [Cremonesi et al., 2008] says, this can lead to an overfitting of the model. Finally, m-fold cross-validation divides a data set into m separate folds and then runs the test m times, each time using a different fold as the test set and the rest combined as the training set.

However, as [Shani and Gunawardana, 2009] emphasises, in order to get an accurate estimation of performance when testing a system offline, one has to mimic the behaviour the system will face when it goes online, including the data available. It is not for our scenario realistic that we have a training set with behavioural data and a set of test data that even after we have made predictions on it we are not allowed to use. In the real world of e-commerce, data becomes available as people do various things on the site. This, among other things, means that from the beginning, you don't have any behavioural data. This simulates the cold start problem. If a method uses only behavioural data, there are a few options of what to do in a cold start scenario. One would be to not give any recommendations and then start recommending the top sellers once you have some behavioural data until your algorithm has enough data to function properly. Another option would be to define a prior probability distribution where all products are equally likely to be recommended and then adjust the probability distribution when more behavioural data becomes available.

[Shani and Gunawardana, 2009] discuss evaluation of recommender algorithms, but not any implementations of them. Apptus has implemented one of the evaluation methods they discuss and describe as ideal, where the time provides the order of the data and all data is test data from the beginning but becomes available for use in the prediction after it has been used as test data.

This approach is beneficial for Apptus' algorithms as it better reflects their ability to adapt to new information and trends but at the same time actually better mimics reality, where behavioural data is seldom divided into static training and testing sets. The only thing you usually have available in the beginning is a set of content data and no behavioural data and this is how Apptus' Test Framework works. You can't 'cheat' by accessing any future information but get it when it appears on the timeline, just like in an online setting.

The Apptus Test Framework will query your algorithm for a set of products to recommend each time an item is clicked in the session data. The number of recommendations for each click can be configured. We decided to generate top-5 recommendations for each click in our tests. During the time the events packets are read and recommendations made, the recommendations and event ids are saved in a file. Once the program has run its course, another program will take the generated recommendation files and analyse them against what actually happened, e.g. which products were bought and which products were clicked and displayed in the recommendation panel. It also calculates precision and recall, among other things.

Apptus also have an algorithm, synergy, that combines results from different recommenders in a smart way (we are not at liberty to explain how it works since it is a company secret). We combine our algorithm with the best individual algorithm they use today within syn-

ergy to see how well our algorithm works as a backfiller, i.e. how much it improves the overall performance. A good backfiller algorithm can also be seen as one whose good recommendations do not overlap much with the other algorithms that are used.

4.4 Baselines

The baselines for this test were made by Björn Brodén at Apptus and are really simple but at the same time manage to use both content based data and behavioural data, which makes the baselines into hybrid recommender systems. The individual baselines work as follows:

Given a click on a product and an attribute, it finds the attribute value/values for that product. Then it uses Apptus' system to find other products with one or several of the values in common and uses behavioural data to get the top-n best sellers among those and returns that.

There is one baseline for every attribute we have used in our clustering and to get comparisons with the backfiller functionality using Apptus' synergy we have used these baselines with Apptus' best individual algorithm within synergy.

4.5 Test Methodology

We decided to do two kinds of tests - short ones and long ones, where the number of packets was the difference between the two kinds of test. The motivation behind this is that the longer tests take a lot longer to run, but some algorithms benefit from more purchase information.

We were provided with data for Company A for a specific time frame and used all the packets in there for the long tests. This turned out to be 32.2 million packets. For the short tests the number of packets was 6.4 times less, i.e., 5 million packets. The reason for using 6.4 times less packets is that we actually set the packet limit for the short tests first and were going to have ten times as many packets in the big test, but there weren't that many packets in the data set. To be able to compare the results, we used the same number of packets for Company B's data set.

The short tests were done first and then nine to ten of the most promising attribute configurations were chosen to do long tests on. We chose to skip some configurations if they were very similar to ones already chosen and they had the same or worse results than them. This was done in favour of trying other configurations because if felt more meaningful to test a diverse set of configurations.

For the short tests, reference tests were run on each individual attribute to get a sense of how much the individual attribute impacted the result, e.g. if clustering was done on Attrribute A followed by subclustering on Attribute B, we wanted to see if the subclustering made any difference or if the result was the same as just clustering on Attribute A.

In choosing attribute combinations, we had to limit our selection as we only had a finite time to run tests on. We decided to only have one and two levels of clustering because we believe that many times the size of the clusters would otherwise be too small to be of much use.

When it came to choosing which attributes to use and how to combine them we arbitrarily chose attribute combinations we thought might be good indicators together. Testing different kinds of weight for an attribute would have introduced even more combinations to test and we did not have that time, so we decided to simply split the weight evenly when clustering more than one attribute on the same level. We also decided to limit each combination to two attributes to minimise the number of configuration variants. Some limits had to be set and these seemed like the best ones.

For some attributes, further choices needed to be made. For attributes with numerical values, one had to choose the number of items wanted in each cluster. For the multi-level clustering, this value was set to 40, 50 and 60 because that seemed like fair sizes to choose products from. For same-level clustering, the number of vector elements that this choice of values created together with some other attributes became too much for the system to handle (i.e. it became extremely slow), so those values were changed to 4000, 5000 and 6000 for same-level clustering.

Other attributes simply had too many values for the system to handle without running out of memory, so for those we had to cap the number of values used. The choice we made here was to both use a figure near the maximum of what the system could handle and some lower figures to compare with to see if it made any difference. From the first data set we learned that this led to too many combinations (for one attribute combination set there were 27 combinations) so for the second data set we decided to just stick with the near maximum figure.

Another choice that had to be made for each configuration was if any filters should be used on how many attribute values an item should have for the item to be included in the first stage of the clustering (only used in the KMeans++-clustering). Since this would complicate things further we decided not to use this feature, except for Category A in the first data set since it had already been tested in the first tests for the specific attributes and found to get good results.

In results for the specific attribute, we will see that the performance of Connector Algorithm 2 as an individual algorithm is much better than the others and because of this reason, only that connector algorithm was used when testing the general algorithm on the first data set. For the second data set a more thorough testing was done where the first three short tests were run on all three connector algorithm and then compared, both with synergy and as individual algorithm, and it found that Connector Algorithm 2 outperformed the other two, so from there on we only used that algorithm.

4.6 Results

In this section we present the results from our tests. First we will present the results from the clustering on two specific attributes on company A's data set and talk about our choices of attributes and algorithms for the tests on the general algorithm. Following that we will present the results on first Company A's set and then on Company B's. Discussion of the results will mainly be done in the next chapter, Discussion, with exception of any parts needed to explain our choices for the test configurations.

4.6.1 Interpreting Test Names

A couple naming conventions have been used in the tables of the results. The use of 'syn' or 'synergy' at the beginning of the name means that this is the result of combining whatever is in the following square brackets with Apptus' best individual algorithm using synergy.

Because of space restrictions in the table all non-essential information has been dropped, including just using A instead of Attribute A and removing the tail explaining which connection algorithm was used unless several of them were used in the tests.

When underscore is used in the result name, that means the clustering was done on two levels and the attribute that is before the underscore was clustered on before the other one. If there is no underscore, clustering was done on one level with all the attributes that are present in the result name. After the attribute name, there may be further specification of filters that were used, such as $FilterOutStopKeepN$. This particular filter means that the N most common tokens were used in the clustering but that first the 20 most common tokens, so called stop words, were filtered out. If no stop words were filtered out the filter will be denoted by $FilterOutNoStopKeepN$. For Attribute E, the $NbrInClusterN$ denotes the number of items per cluster that were specified to generate the bucket definition; see B How To Configure the Conditions File. Most of the other filters are pretty self-explanatory.

4.6.2 Results from Attribute Specific Clustering

In Table 4.1, we present the results from the short test run for specific attributes (Attribute A and Attribute E) on company A's data set. In Table 4.2, the result is shown for the long test on the same attributes. When it comes to individual algorithm results, we see that there is a shift in how well they do compared to each other in the short and long tests. In the longer test the system with connection algorithm 2 is no longer about as good as the system with connection algorithm 3 but rather a lot better than both of the others. Since the system is intended to be used a longer time we have chosen to place more importance in the results from the longer run as we feel this gives a more accurate description of that behaviour.

When looking at the results from the long tests using Apptus' synergy algorithm we see that the recall and precision figures are pretty close to each other. For this reason and

Algorithm Name	Recall	Precision	Purchases
Alg1	0.008	0.009	27[48.64%, 21.62%, 29.72%, 0.0%, 0.0%]
Alg2	0.015	0.025	46[30.98%, 26.76%, 29.57%, 9.85%, 2.81%]
Alg3	0.016	0.02	50[24.09%, 30.12%, 24.09%, 15.66%, 6.02%]
synergy[alg2]	0.024	0.039	75[36.36%, 30.9%, 13.63%, 9.09%, 10.0%]
synergy[baseline]	0.025	0.028	79[36.97%, 23.52%, 11.76%, 13.44%, 14.28%]
synergy[alg1]	0.026	0.029	81[29.2%, 27.43%, 22.12%, 9.73%, 11.5%]
synergy[alg3]	0.028	0.034	87[27.94%, 27.2%, 21.32%, 12.5%, 11.02%]

Table 4.1: Results from the short tests run for specific attributes (Attribute A and Attribute E) on Company A's data set.

Algorithm Name	Recall	Precision	Purchases
Alg3	0.013	0.019	586[44.02%, 21.51%, 16.66%, 10.69%, 7.08%]
Alg1	0.015	0.02	662[43.38%, 22.84%, 13.44%, 10.81%, 9.5%]
Alg2	0.026	0.036	1106[29.92%, 22.92%, 18.85%, 14.54%, 13.76%]
synergy[baseline]	0.052	0.064	2214[36.18%, 22.77%, 16.6%, 12.92%, 11.51%]
synergy[alg2]	0.052	0.069	2226[36.94%, 22.83%, 16.47%, 12.13%, 11.61%]
synergy[alg3]	0.053	0.07	2250[36.44%, 23.75%, 16.5%, 12.29%, 11.0%]
synergy[alg1]	0.053	0.066	2277[36.24%, 23.45%, 16.16%, 12.52%, 11.6%]

Table 4.2: Results from the long tests run for specific attributes (Attribute A and Attribute E) on Company A's data set.

because connection algorithm 2 performed so much better individually than the other two, we chose to disregard the more marked difference in number of purchases when using the synergy algorithm and decided to do further tests in Company A's data set using the system with connection algorithm 2. The reason why we did not decide to use all the connector algorithms in the next step of testing was because it would have taken too much time.

An important note in comparison with the tests from the general implementation is that here items are only filtered out to be used in the initial clustering if they have at least 3 values of a specific type for Attribute A, compared to later tests where this was often instead at least 2 values. How many values that were required will be made clear in each result section.

4.6.3 Results for Company A

In this section we will present the results of our tests on our general implementation using Company A's data set. For confidentiality reasons the names of the attributes we have used had to be encoded so they are in this report referred to as Attributes A, B, C, D and E. All those attributes except Attribute E are categorical in nature. Attribute E is a numerical attribute.

Short Tests

For the smaller number of packages, we ran a total of 72 tests for our own algorithms, including the reference tests. In order not to clutter up the chapter with too many tables, the ones with the results from the short tests have been put in Appendix A.

Long Tests

For the longer tests we decided to choose 10 of the most promising algorithms from the short tests. We didn't choose the ten top results straight off since some of the top combinations are pretty similar and we wanted to give the other combinations a chance to get a better score on the longer tests. In Table 4.4, the results for our chosen combinations are shown. The corresponding results for the baselines are shown in Table 4.5.

Just as for the short tests the results for our own algorithms and the baselines were separately combined with Apptus' best individual algorithm using synergy and the results are shown in Tables 4.6 and 4.7 respectively. Apptus' best individual algorithm had a recall of 0.08, precision value of 0.152 and 2188 purchases for the long test.

Algorithm Name	Recall	Precision	Purchases
D:FilterOutStopKeep2000E:NbrInCluster5000	0.027	0.04	747[26.63%, 18.92%, 20.54%, 16.94%, 16.94%]
C:D:FilterOutStopKeep2000	0.03	0.053	819[26.19%, 21.99%, 19.58%, 18.23%, 13.99%]
A:Filter2ValuesB:Freq5000	0.036	0.047	991[24.98%, 20.49%, 20.79%, 18.05%, 15.66%]
B:Freq5000_A:2Values	0.037	0.049	1026[27.33%, 19.86%, 17.42%, 17.56%, 17.8%]
E:NbrInCluster60_D:FilterOutStopKeep1000	0.037	0.047	1030[28.2%, 22.18%, 17.58%, 16.69%, 15.33%]
A:Filter2Values	0.038	0.055	1058[25.47%, 19.85%, 20.49%, 17.19%, 16.98%]
A:Filter2ValuesE:NbrInCluster4000	0.039	0.052	1068[25.52%, 18.81%, 18.15%, 16.06%, 21.43%]
B:Freq5000_A:	0.04	0.048	1098[29.25%, 21.28%, 17.17%, 16.14%, 16.14%]
A:Filter2Values_E:NbrInCluster60	0.044	0.052	1218[29.89%, 21.46%, 19.05%, 15.78%, 13.79%]
A:Filter2Values_E:NbrInCluster50	0.045	0.053	1229[27.86%, 22.64%, 16.56%, 17.16%, 15.76%]

Table 4.4: Results from the long tests run for the best combinations of our own general algorithm on Company A's data set.

Algorithm Name	Recall	Precision	Purchases
D:_exact	0.004	0.028	118[87.58%, 9.65%, 2.06%, 0.68%, 0.0%]
topSellers	0.017	0.048	488[22.64%, 23.82%, 16.5%, 17.29%, 19.73%]
D:	0.024	0.034	674[35.95%, 23.65%, 14.66%, 14.25%, 11.46%]
E:	0.025	0.036	694[30.96%, 23.77%, 17.48%, 13.48%, 14.28%]
A:	0.033	0.065	915[31.13%, 16.85%, 16.26%, 17.25%, 18.49%]
B:	0.034	0.047	924[33.38%, 22.88%, 17.88%, 12.67%, 13.17%]
C:	0.047	0.086	1297[28.09%, 25.65%, 18.03%, 14.96%, 13.25%]
B:_exact	0.078	0.136	2129[34.02%, 24.05%, 16.6%, 14.6%, 10.7%]

Table 4.5: Results from the long tests run for the baselines on Company A's data set.

Algorithm Name	Recall	Precision	Purchases
syn[E:NbrInCluster60_D:FilterOutStopKeep1000]	0.083	0.092	2268[36.28%, 23.21%, 16.23%, 12.35%, 11.9%]
syn[A:Filter2Values_E:NbrInCluster50]	0.083	0.09	2270[36.14%, 23.28%, 16.27%, 12.47%, 11.82%]
syn[A:Filter2Values_E:NbrInCluster60]	0.084	0.09	2288[36.24%, 23.14%, 16.25%, 12.74%, 11.61%]
syn[D:FilterOutStopKeep2000E:NbrInCluster5000]	0.085	0.084	2325[35.26%, 23.47%, 16.39%, 12.67%, 12.2%]
syn[A:Filter2ValuesB:Freq5000]	0.085	0.083	2332[35.47%, 22.8%, 16.72%, 12.78%, 12.21%]
syn[B:Freq5000_A:]	0.086	0.082	2336[36.33%, 22.88%, 16.36%, 12.39%, 12.02%]
syn[C:D:FilterOutStopKeep2000]	0.086	0.087	2345[35.52%, 22.85%, 17.02%, 12.58%, 12.01%]
syn[B:Freq5000_A:2Values]	0.086	0.084	2348[35.98%, 23.13%, 15.95%, 12.54%, 12.37%]
syn[A:Filter2ValuesE:NbrInCluster4000]	0.086	0.085	2360[35.15%, 22.31%, 16.21%, 13.09%, 13.22%]
syn[A:Filter2Values]	0.087	0.086	2373[35.06%, 23.23%, 16.66%, 12.77%, 12.26%]

Table 4.6: Results from the long tests run for the best combinations of our own general algorithm on Company A's data set combined with Apptus' best individual algorithm using synergy.

Algorithm Name	Recall	Precision	Purchases
syn[AllBaselines]	0.08	0.152	2188[36.61%, 23.39%, 16.19%, 12.16%, 11.63%]
syn[D:_exact]	0.082	0.146	2227[36.85%, 23.63%, 16.29%, 11.62%, 11.58%]
syn[E:]	0.084	0.101	2283[36.29%, 23.24%, 16.34%, 12.36%, 11.74%]
syn[topSellers]	0.084	0.108	2305[34.87%, 23.93%, 16.32%, 12.5%, 12.36%]
syn[B:]	0.086	0.109	2362[35.97%, 23.09%, 16.24%, 12.68%, 11.99%]
syn[A:]	0.088	0.111	2413[35.97%, 23.12%, 16.21%, 12.57%, 12.1%]
syn[D:]	0.089	0.111	2419[35.62%, 23.95%, 16.19%, 12.56%, 11.66%]
syn[C:]	0.096	0.123	2618[35.22%, 24.37%, 16.53%, 12.77%, 11.1%]
syn[B:_exact]	0.102	0.147	2772[35.21%, 23.88%, 16.12%, 13.65%, 11.11%]

Table 4.7: Results from the long tests run for the baselines on Company A's data set combined with Apptus' best individual algorithm using synergy.

4.6.4 Results for Company B

Since we did not have any indications specific for Company B's data set on which connection algorithm would be most beneficial to test on we decided to do three short tests for each connection algorithm using different attribute combinations. The results showed great promise for using the system as an individual algorithm as there were combinations that managed to beat their baselines and so we focused on the individual algorithm results when deciding which connection algorithm to go forward with.

Short Tests

Just as for Company A's data set, the results from the short tests are presented in Appendix A. The combinations with connection algorithm 2 outperformed both those with connection algorithm 1 and connection algorithm 3 when looking at individual results so we decided to use that one for the long tests.

Long Tests

Just as for Company A's data set, we decided to choose 10 of the most promising combinations rom the short tests. The results from running the longer test on these combinations is shown in Table 4.8. The results of the long tests on the baselines are presented in Table 4.9.

Algorithm Name	Recall	Precision	Purchases
G:Freq5000A:_MAH2	0.146	0.082	4542[27.18%, 22.05%, 20.26%, 16.53%, 13.96%]
G:Freq5000_A:_MAH2	0.146	0.083	4550[27.16%, 22.0%, 20.15%, 16.49%, 14.19%]
I:_A:_MAH2	0.146	0.083	4555[27.17%, 22.05%, 20.2%, 16.42%, 14.13%]
H:_A:_MAH2	0.146	0.083	4556[27.18%, 22.05%, 20.2%, 16.41%, 14.13%]
A:J:_MAH2	0.148	0.074	4614[27.57%, 21.58%, 19.19%, 16.99%, 14.64%]
F:A:_MAH2	0.149	0.081	4651[26.54%, 22.09%, 19.46%, 16.83%, 15.05%]
A:H:_MAH2	0.156	0.08	4849[26.25%, 20.14%, 20.05%, 17.85%, 15.69%]
A:_H:_MAH2	0.157	0.08	4906[26.66%, 21.43%, 19.76%, 16.9%, 15.24%]
F:_A:_MAH2	0.179	0.082	5575[31.28%, 21.86%, 18.28%, 15.61%, 12.93%]
A:_F:_MAH2	0.18	0.082	5607[30.63%, 21.84%, 18.22%, 16.27%, 13.02%]

Table 4.8: Results from the long tests run for our own general algorithm on Company B's data set.

Algorithm Name	Recall	Precision	Purchases
G:	0.0	0.001	9[66.66%, 11.11%, 0.0%, 11.11%, 11.11%]
A:	0.164	0.088	847[34.33%, 24.56%, 14.33%, 14.85%, 11.9%]
Topsellers	0.059	0.083	1838[21.38%, 22.39%, 20.39%, 17.42%, 18.39%]
H:	0.061	0.076	1914[21.78%, 23.26%, 19.49%, 19.66%, 15.78%]
I:	0.077	0.05	2406[31.35%, 21.8%, 17.66%, 16.37%, 12.79%]
J:	0.091	0.065	2853[27.24%, 21.98%, 19.55%, 16.66%, 14.55%]
F:	0.129	0.067	4014[29.4%, 21.43%, 19.14%, 16.0%, 14.01%]

Table 4.9: Results from the long tests run for the baselines on Company B's data set.

Like for all the other tests, we combined our own results with Apptus' individual algorithm using synergy and the results of that are shown in Table 4.10. The corresponding results for the baselines are presented in Table 4.11. Apptus' best individual algorithm had a recall of 0.211, precision value of 0.082 and 6564 purchases for the long test.

Algorithm Name	Recall	Precision	Purchases
syn[F:A:_MAH2]	0.177	0.085	5505[27.01%, 22.48%, 19.06%, 16.28%, 15.15%]
syn[G:Freq5000_A:_MAH2]	0.181	0.088	5650[26.4%, 21.48%, 21.03%, 16.35%, 14.71%]
syn[H:_A:_MAH2]	0.182	0.088	5660[26.54%, 21.47%, 20.9%, 16.6%, 14.47%]
syn[I:_A:_MAH2]	0.182	0.088	5669[26.41%, 21.46%, 21.07%, 16.62%, 14.42%]
syn[G:Freq5000A:_MAH2]	0.186	0.089	5789[26.36%, 22.64%, 19.5%, 16.48%, 14.99%]
syn[A:_F:_MAH2]	0.187	0.081	5818[30.23%, 21.74%, 18.36%, 16.24%, 13.4%]
syn[A:J:_MAH2]	0.189	0.082	5894[27.13%, 21.15%, 17.69%, 16.03%, 17.98%]
syn[A:_H:_MAH2]	0.199	0.085	6190[27.22%, 22.16%, 19.08%, 15.76%, 15.75%]
syn[F:_A:_MAH2]	0.2	0.084	6222[29.86%, 20.99%, 17.72%, 15.33%, 16.07%]
syn[A:H:_MAH2]	0.204	0.085	6339[26.63%, 23.61%, 18.73%, 15.55%, 15.45%]

Table 4.10: Results from the long tests run for the best combinations of our own general algorithm on Company B's data set combined with Apptus' best individual algorithm using synergy.

Algorithm Name	Recall	Precision	Purchases
syn[H:]	0.178	0.087	5559[26.65%, 21.48%, 18.05%, 16.8%, 17.0%]
syn[J:]	0.193	0.083	6024[28.54%, 21.75%, 18.0%, 16.23%, 15.46%]
syn[F:]	0.203	0.081	6319[29.08%, 22.29%, 18.43%, 15.45%, 14.72%]
syn[I:]	0.209	0.081	6523[28.21%, 23.75%, 18.34%, 15.23%, 14.44%]
syn[A:]	0.21	0.084	6554[28.12%, 23.0%, 19.02%, 15.99%, 13.85%]
syn[G:]	0.211	0.082	6564[29.34%, 22.57%, 18.55%, 15.38%, 14.13%]

Table 4.11: Results from the long tests run for the baselines on Company B's data set combined with Apptus' best individual algorithm using synergy.

Chapter 5
Discussion

In this chapter we will first discuss the results presented in the last chapter and look at possible sources of error. We will also briefly describe the positive and negative experiences we have had while working on this thesis before we take a look at possible future work related to the results and sources of error.

5.1 Discussion of Results

Our results from our early test on Company A's data set using Attributes A and E were promising since all three algorithms managed to beat the baseline using synergy. This indicated that our system might work well as a backfiller. However, once we went to test the general system, things did not go nearly as well on Company A's data set. As can be seen when comparing the individual results in Tables A.1 and A.2, the best baseline is leaps and bounds ahead of any of our algorithms for the short tests. The same is true for the short tests using synergy, as can be seen when comparing Tables A.3 and A.4. This means for the short tests there is no indication that our algorithm would work well either as an individual algorithm or as a backfiller. Unfortunately, one can see that the same holds true when running the tests for a longer time by comparing Tables 4.4 and 4.5, for individual algorithm results, and Tables 4.6 and 4.7, for the backfiller functionality. The difference is not as big when comparing backfiller functionality as when using them as individual algorithms but it is still clear that the baselines are the better option result wise. This and the principle of Occam's Razor[1] tells us it would have been a good idea to start with the baselines and work on those instead of doing clustering. That wouldn't have been nearly as interesting, though, in our opinion. Another thing to note is that the best baseline performs pretty well as an individual algorithm as well, even if Apptus' best individual algorithm is better. However, Apptus' best individual algorithm has a bit of an unfair advantage that we will talk more about when discussing the results on Company B's data set.

[1] https://en.wikipedia.org/wiki/Occam's_razor

After the results from Company A's data set we were pleasantly surprised to note that most of the combinations of our algorithm and most of the references got better results than the corresponding baselines when looking at the individual algorithms. Most importantly, as can be seen when comparing Tables A.4 and A.5, the best individual algorithm was one of the combinations for our algorithm. When using our algorithm as a backfiller the results were not as good, which can be seen when comparing Tables A.6 and A.7. Here the best result with synergy is actually achieved by one of the baselines (the one with Attribute A) which indicates that for Company B's data set our algorithm would work best as an individual algorithm rather than as a backfiller. Comparing the results in Tables 4.8 and 4.9 one can see that the results for the individual algorithms are very similar to the short ones. A comparison of Tables 4.10 and 4.11 shows that this holds true for the backfiller functionality as well.

Now, this looks like very good results, but we have neglected to take into account the results for Apptus' own best individual algorithm. Comparing our best combination for the short tests with Apptus' best individual algorithm shows that ours has more than 100 purchases less than it, which means it has about 12% more purchases than our best algorithm. For the long tests, it has slightly less than 1000 purchases more than our best combination. This translates to about 17% more purchases than our best algorithm. Clearly, Apptus' best individual algorithm is increasing its results relative to our best algorithm over time.

An important thing to note here is that these tests are only offline tests. That means it will only measure how good your system is at recommending the products the customers actually bought without your recommendations influencing them. In an online setting, using e.g. an A/B test, your recommendations might influence the customer and cause them to buy things they did not in the offline test. On the other hand, it would also cause the customers to not see the other recommendations they did in the offline test and that might cause them to buy less products or different products altogether.

Another important thing that we did not think of when we began testing is that Apptus' best individual algorithm (or at least an earlier version of it) was used online in the time period Company B's data set was generated from. This means that the algorithm has given recommendations to the customers and might have influenced their purchases. We thinks this is a very unfair advantage and introduces a bias into the data set that favours Apptus' algorithm. There is no way to know how much the result for Apptus' algorithm would have been affected if we had managed to find data where Apptus' algorithm had not been active online, but the online factor should definitely be taken into account when comparing the results.

As we established earlier in the discussion, our algorithm does not seem cut out to work as a backfiller to Apptus' best individual algorithm. This is because instead of complementing each other's recommendations there is probably a big overlap in the recommendations they make. A strong algorithm by itself is not always the best choice since an ensemble of not-so-good recommenders could be preferable if they complement each other and provide a better result than one individual algorithm when they work together. We think the

choices of algorithms used to work together and the combining algorithm (which in our case was synergy) can make a big impact on the performance of the overall system.

We have so far not talked about any comparisons between the different combinations or talked any more about the results from the different connection algorithms than what we did in the Results section. Looking at the synergy results for Company A's data set we can see that there is movement in the top results when comparing the results from the short and long tests using synergy. What seems consistent between the results is that clusterings that use Attribute A or B on the first level do well. For the baselines, using the exact value of Attribute B (i.e. without tokenising the value), gives the best results for the shorter and longer tests on Company A's data set.

The best results from the individual algorithms for the short tests on Company B's data set seems to consist of combinations of Attributes A, F and H. Attribute F was also the best attribute when using the baselines on the short tests. For the long tests on Company B's data the same attributes seem to give the best results as well and for the baselines Attribute F is again the best. The reason why we think Attribute A is doing well in both data sets is that it has a tree-like structure where there is a lot of information that can be used when clustering the products. We have no thoughts we can write down on why Attributes F and H are good indicators for the Company B's data set without revealing too much about the attributes and so cannot hold any discussion about these. Attribute B from Company A's data set can be chopped into several values and we think that is part of the reason why we got good results on the combinations that used Attribute B to cluster on in the first level. This allowed us to find connections between items that did not have the same but similar values. We might also have gotten even better results if we had not had to restrict the number of values used due to the design of the general implementation.

During the work of this thesis many critical choices had to be made and thus there are several sources of error that might have affected the results we have gotten. These are of two different natures - choices pertaining to the whole system with one clustering algorithm system and one connection algorithm and choices pertaining to clustering with kMeans++. For the whole system we think the main sources of error are which clustering algorithm to use, how many attributes should be used on one level of clustering and also how the levels of clustering is determined. One thing that could have been done differently for the levels is to have a more dynamic model where the number of products in a cluster determines if the products from there should be clustered further.

There are many sources of error that are connected to the use of the kMeans++ algorithm and our general implementation where we use it. The first one is the value of k since using different k's can lead to vastly different groupings of points, or items in our case. The second source of error we feel is worth mentioning is the attribute weights. To limit the number of combinations when we had several attributes on one level we had to decide on one attribute weight combination and ended up splitting the weight evenly. With other weights we would have gotten different results (who knows if they would have been better or worse) but we felt splitting it evenly was the best way to go considering we would have the clustering on two levels and the reference clusterings to compare the results with. The

last and third source of error we think worth mentioning is the choice and implementation of transforms for the attributes, i.e., the way the attribute values are transformed from their actual values to the double array used by Apache's implementation of the KMeans++ algorithm. For some attributes information is lost because there are too many possible values to take into account and so not all of them will be used in the array.

There have been many new experiences during the work of this thesis. We have had the chance to work at a company who deploys a recommender system to customers and work in their software environment. We have also worked in a research team and both had help from them and had to make compromises to make the work fit with the purposes and workings of the research group.

In hindsight, it is easier to see the mistakes done along the way and one thing we definitely would have done different is to spend more time in the beginning researching different clustering methods instead of concentrating on the ones we already knew of. The fact that we during the report writing have seen many of the mistakes we made along the way is to us a sign of how much we have learned. We came into this project with a few courses of knowledge about AI and data mining but with hardly any knowledge of recommender systems. We feel that we have learned a lot during the work of this thesis, especially about the field of recommender systems but also about data mining.

5.2 Future Work

As mentioned earlier, the use of another clustering method more suited to clustering mixed kinds of data, e.g. kModes, might give better results and could be worth investigating. Also, an important point would be to try and implement a system with fewer sources of error and that requires less manual testing.

Another option would be to implement a system that could get rid of some sources of error like the weights for the attributes through automated testing as this is otherwise something that would have to be done manually for each of Apptus' clients.

One interesting thing that came from the results was that Attribute A seemed to give good clusterings for both data sets. It might be worth it to further investigate if Attribute A is a generally good clusterer for data sets and if the same is true for attributes with a tree-like structures in general.

More testing using different connection algorithms might also be warranted since that mostly had to be skipped in this thesis due to time constraints. It would also be interesting to see if a system using our algorithm with the baselines as backfillers would improve the systems performance on Company B's data set.

Chapter 6
Conclusions

For the earlier testing using specific attributes from Company A's data set the system looked promising to use as a backfiller. However, further testing on Company A's data set using the general system implementation belied this and did not introduce any potential for it to act as a strong individual algorithm either.

For Company B's data set our algorithm showed big potential to work as an individual algorithm but it was outperformed by Apptus' best individual algorithm. Something to keep in mind however is that it had an unfair advantage (see section 5.1 Discussion of Results). It did not show much potential for working as a backfiller here since combining Apptus' best individual algorithm with one of the baselines gave better results for both the short and long tests.

This makes two things in particular clear:

1. The algorithm will not have the same functionality (i.e. backfiller or individual algorithm) on all data sets
2. A strong individual algorithm does not necessarily give a better overall result when combining it with another strong algorithm

While the difference in which functionality our algorithm plays for different data sets does not matter that much, the fact that it only gives good results on one of them does, since the aim of our thesis was for it to work on e-commerce sites in general and not just a specific one.

However, we still think it would be of interest to do further research on hybrid recommenders using more appropriate clustering methods and comparing them with other algorithms on data sets with no bias towards one of the algorithms.

Bibliography

[Adomavicius and Tuzhilin, 2005] Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749.

[Aldrich, 2015] Aldrich, S. E. (2015). Recommender systems in commercial use. *AI Magazine*, 32(3):28–34.

[Arthur and Vassilvitskii, 2007] Arthur, D. and Vassilvitskii, S. (2007). K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

[Balabanović and Shoham, 1997] Balabanović, M. and Shoham, Y. (1997). Fab: Content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72.

[Bell et al., 2010] Bell, R. M., Koren, Y., and Volinsky, C. (2010). All together now: A perspective on the netflix prize. *Chance*, 23(1):24–29.

[Burke, 2002] Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370.

[Cremonesi et al., 2008] Cremonesi, P., Turrin, R., Lentini, E., and Matteucci, M. (2008). An evaluation methodology for collaborative recommender systems. In *Proceedings of the 2008 International Conference on Automated Solutions for Cross Media Content and Multi-channel Distribution*, AXMEDIS '08, pages 224–231, Washington, DC, USA. IEEE Computer Society.

[Gantner et al., 2010] Gantner, Z., Drumond, L., Freudenthaler, C., Rendle, S., and Schmidt-Thieme, L. (2010). Learning attribute-to-feature mappings for cold-start recommendations. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 176–185, Washington, DC, USA. IEEE Computer Society.

[Hu et al., 2008] Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *In IEEE International Conference on Data Mining (ICDM 2008*, pages 263–272.

[Huang, 1998] Huang, Z. (1998). Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2(3):283–304.

[Jiang et al., 2013] Jiang, X., Niu, Z., Guo, J., Mustafa, G., Lin, Z.-H., Chen, B., and Zhou, Q. (2013). Novel boosting frameworks to improve the performance of collaborative filtering. In *ACML*, pages 87–99.

[Kim et al., 2006] Kim, B. M., Li, Q., Park, C. S., Kim, S. G., and Kim, J. Y. (2006). A new approach for combining content-based and collaborative filters. *J. Intell. Inf. Syst.*, 27(1):79–91.

[Leander, 2014] Leander, M. (2014). Category recommendations in e-commerce systems. Master's thesis, Lund University.

[Linden et al., 2003] Linden, G., Smith, B., and York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80.

[Lops et al., 2009] Lops, P., de Gemmis, M., and Semeraro, G. (2009). Content-based recommender systems : State of the art and trends. In Kantor, P., Ricci, F., Rokach, L., and Shapira, B., editors, *Recommender Systems Handbook*. Springer. to appear.

[Mardia et al., 1979] Mardia, K. V., Kent, J. T., and Bibby, J. M. (1979). *Multivariate Analysis*. Academic Press. pg. 365.

[Meyer, 2012] Meyer, F. (2012). *Recommender systems in industrial contexts*. PhD thesis, l'Université de Grenoble.

[Nageswara and Talwar, 2008] Nageswara, R. K. and Talwar, V. G. (2008). Application domain and functional classification of recommender systems – a survey. *DESIDOC Journal of Library & Information Technology*, 28(3):17–35. 28(3):17-35.

[Ning and Karypis, 2012] Ning, X. and Karypis, G. (2012). Sparse linear methods with side information for top-n recommendations. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 155–162, New York, NY, USA. ACM.

[Oard and Kim, 1998] Oard, D. W. and Kim, J. (1998). Implicit feedback for recommender systems. In *Proceedings of the AAAI Workshop on Recommender Systems*.

[Ostuni et al., 2013] Ostuni, V. C., Di Noia, T., Di Sciascio, E., and Mirizzi, R. (2013). Top-n recommendations from implicit feedback leveraging linked open data. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 85–92, New York, NY, USA. ACM.

[Peska and Vojtas, 2014] Peska, L. and Vojtas, P. (2014). *Recommending for disloyal customers with low consumption rate.*, volume 8327 LNCS of *Lecture Notes in Computer Science*. Faculty of Mathematics and Physics, Charles University in Prague.

[Rendle and Freudenthaler, 2014] Rendle, S. and Freudenthaler, C. (2014). Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 273–282, New York, NY, USA. ACM.

[Rendle et al., 2009] Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. (2009). Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 452–461, Arlington, Virginia, United States. AUAI Press.

[Sarwar et al., 2001] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA. ACM.

[Shani and Gunawardana, 2009] Shani, G. and Gunawardana, A. (2009). Evaluating recommender systems. Technical Report MSR-TR-2009-159.

[Su and Khoshgoftaar, 2009] Su, X. and Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2.

[Töscher et al., 2008] Töscher, A., Jahrer, M., and Legenstein, R. (2008). Improved neighborhood-based algorithms for large-scale recommender systems. In *Proceedings of the 2Nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*, NETFLIX '08, pages 4:1–4:6, New York, NY, USA. ACM.

[Tso and Schmidt-Thieme, 2006] Tso, K. and Schmidt-Thieme, L. (2006). Attribute-aware collaborative filtering. In Spiliopoulou, M., Kruse, R., Borgelt, C., N??rnberger, A., and Gaul, W., editors, *From Data and Information Analysis to Knowledge Engineering*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 614–621. Springer Berlin Heidelberg.

[Wu, 2007] Wu, M. (2007). Collaborative filtering via ensembles of matrix factorizations. In *KDD Cup and Workshop 2007*, pages 43–47. Max-Planck-Gesellschaft.

Appendices

Appendix A
The Rest of the Result Tables

This appendix contains the rest of the result tables that were not included in the Results chapter to limit the amount of tables there. In choosing which tables to move here we decided to move the tables from the short tests here, partly because they took up the more space than those from the long tests and partly because the results from long tests are more important in our opinion.

A.1 Results for Company A

In this section the results from the short test on Company A's data set will be presented. First the results from the individual algorithms will be presented and then the results when combining the algorithms with Apptus' best individual algorithm using Synergy will be presented.

A.1.1 Individual Algorithm Results

For the smaller number of packages, we ran a total of 72 tests for our own algorithms, including the reference tests. In Table A.1, you can see the results from our own algorithms (including the reference tests) and in Table A.2, the results of the baseline algorithm for the different attributes are shown.

Algorithm Name	Recall	Precision	Purchases
A:Filter2Values_B:Top5000	0.002	0.266	5[33.33%, 50.0%, 8.33%, 8.33%, 0.0%]
D:FilterOutStopKeep2000	0.011	0.025	22[6.59%, 17.58%, 27.47%, 38.46%, 9.89%]
B:Top1000	0.011	0.025	22[6.59%, 16.48%, 20.87%, 39.56%, 16.48%]
D:FilterOutStopKeep4000	0.011	0.025	22[6.59%, 17.58%, 27.47%, 38.46%, 9.89%]
D:FilterOutStopKeep1000	0.011	0.025	22[6.59%, 17.58%, 27.47%, 38.46%, 9.89%]
B:Freq5000	0.011	0.025	22[6.59%, 17.58%, 27.47%, 38.46%, 9.89%]

Algorithm Name	Recall	Precision	Purchases
E:NbrInCluster60_A:Filter2Values	0.011	0.026	23[33.33%, 11.11%, 22.22%, 19.44%, 13.88%]
E:NbrInCluster40_D:FilterOutStopKeep1000	0.012	0.018	24[33.33%, 15.15%, 24.24%, 15.15%, 12.12%]
D:FilterOutStopKeep4000E:NbrInCluster4000	0.012	0.029	24[5.82%, 11.65%, 14.56%, 24.27%, 43.68%]
E:NbrInCluster4000	0.012	0.015	25[30.0%, 30.0%, 17.5%, 10.0%, 12.5%]
E:NbrInCluster6000	0.012	0.018	25[19.14%, 31.91%, 27.65%, 8.51%, 12.76%]
D:FilterOutStopKeep1000_E:NbrInCluster50	0.013	0.017	27[33.33%, 19.44%, 25.0%, 16.66%, 5.55%]
E:NbrInCluster60	0.014	0.017	28[35.13%, 18.91%, 24.32%, 16.21%, 5.4%]
E:NbrInCluster60_D:FilterOutStopKeep2000	0.014	0.019	28[35.13%, 21.62%, 24.32%, 16.21%, 2.7%]
D:FilterOutStopKeep2000_E:NbrInCluster40	0.014	0.018	28[37.83%, 16.21%, 24.32%, 16.21%, 5.4%]
D:FilterOutStopKeep4000_C:	0.014	0.017	28[28.33%, 25.0%, 23.33%, 5.0%, 18.33%]
D:FilterOutStopKeep4000_E:NbrInCluster50	0.014	0.018	28[35.13%, 18.91%, 24.32%, 16.21%, 5.4%]
E:NbrInCluster40_D:FilterOutNoStopKeep4000	0.014	0.021	28[31.57%, 18.42%, 18.42%, 21.05%, 10.52%]
D:FilterOutStopKeep1000_E:NbrInCluster40	0.014	0.018	28[37.83%, 16.21%, 24.32%, 16.21%, 5.4%]
E:NbrInCluster50	0.014	0.017	28[35.13%, 18.91%, 24.32%, 16.21%, 5.4%]
D:FilterOutNoStopKeep4000_E:NbrInCluster40	0.014	0.018	28[37.83%, 16.21%, 24.32%, 16.21%, 5.4%]
D:FilterOutStopKeep2000_E:NbrInCluster60	0.014	0.017	28[35.13%, 18.91%, 24.32%, 16.21%, 5.4%]
D:FilterOutStopKeep1000_E:NbrInCluster60	0.014	0.017	28[35.13%, 18.91%, 24.32%, 16.21%, 5.4%]
D:FilterOutStopKeep2000_E:NbrInCluster50	0.014	0.018	28[35.13%, 18.91%, 24.32%, 16.21%, 5.4%]
E:NbrInCluster40	0.014	0.018	28[37.83%, 16.21%, 24.32%, 16.21%, 5.4%]
D:FilterOutStopKeep4000_E:NbrInCluster40	0.014	0.018	28[37.83%, 16.21%, 24.32%, 16.21%, 5.4%]
D:FilterOutStopKeep4000_E:NbrInCluster60	0.014	0.017	28[35.13%, 18.91%, 24.32%, 16.21%, 5.4%]
E:NbrInCluster50_D:FilterOutStopKeep1000	0.014	0.02	28[24.32%, 29.72%, 16.21%, 24.32%, 5.4%]
D:FilterOutStopKeep2000_C:	0.014	0.015	28[22.64%, 33.96%, 24.52%, 11.32%, 7.54%]
E:NbrInCluster50_A:Filter2Values	0.014	0.02	29[34.21%, 21.05%, 28.94%, 10.52%, 5.26%]
E:NbrInCluster60_D:FilterOutStopKeep4000	0.014	0.02	29[34.21%, 23.68%, 15.78%, 15.78%, 10.52%]
E:NbrInCluster40_D:FilterOutStopKeep2000	0.014	0.023	29[39.53%, 30.23%, 11.62%, 13.95%, 4.65%]
D:FilterOutStopKeep1000E:NbrInCluster4000	0.014	0.014	29[31.37%, 31.37%, 13.72%, 9.8%, 13.72%]
C:D:FilterOutStopKeep1000	0.015	0.02	30[23.28%, 23.28%, 5.47%, 12.32%, 35.61%]
D:FilterOutStopKeep2000E:NbrInCluster6000	0.015	0.015	30[20.75%, 32.07%, 28.3%, 7.54%, 11.32%]
E:NbrInCluster50_D:FilterOutStopKeep2000	0.015	0.022	30[29.26%, 17.07%, 21.95%, 9.75%, 21.95%]
D:FilterOutStopKeep1000E:NbrInCluster6000	0.015	0.015	30[20.75%, 32.07%, 28.3%, 7.54%, 11.32%]
C:D:FilterOutStopKeep4000	0.015	0.018	31[28.12%, 26.56%, 17.18%, 7.81%, 20.31%]
D:FilterOutStopKeep4000E:NbrInCluster6000	0.015	0.025	31[17.77%, 14.44%, 13.33%, 45.55%, 8.88%]
C:_D:FilterOutStopKeep2000	0.015	0.021	31[26.38%, 20.83%, 25.0%, 6.94%, 20.83%]
C:_D:FilterOutStopKeep4000	0.015	0.021	31[23.61%, 30.55%, 9.72%, 15.27%, 20.83%]
C:	0.016	0.018	32[23.43%, 34.37%, 15.62%, 9.37%, 17.18%]
E:NbrInCluster50_D:FilterOutStopKeep4000	0.016	0.024	32[23.91%, 23.91%, 19.56%, 17.39%, 15.21%]
D:FilterOutStopKeep1000E:NbrInCluster5000	0.016	0.015	32[24.07%, 12.96%, 29.62%, 22.22%, 11.11%]
D:FilterOutStopKeep4000E:NbrInCluster5000	0.016	0.018	32[16.66%, 27.27%, 19.69%, 16.66%, 19.69%]
E:NbrInCluster5000	0.016	0.021	32[24.07%, 12.96%, 29.62%, 22.22%, 11.11%]
D:FilterOutStopKeep1000_C:	0.016	0.029	33[8.65%, 20.19%, 18.26%, 33.65%, 19.23%]
A:Filter2Values_E:NbrInCluster40	0.016	0.031	33[24.44%, 33.33%, 15.55%, 20.0%, 6.66%]
E:NbrInCluster40_D:FilterOutStopKeep4000	0.016	0.025	33[31.11%, 26.66%, 13.33%, 24.44%, 4.44%]
E:NbrInCluster60_D:FilterOutStopKeep1000	0.016	0.024	33[35.55%, 20.0%, 15.55%, 20.0%, 8.88%]
C:_D:FilterOutStopKeep1000	0.017	0.02	34[22.22%, 20.83%, 13.88%, 8.33%, 34.72%]
E:NbrInCluster5000_D:FilterOutStopKeep2000	0.017	0.022	34[22.8%, 12.28%, 28.07%, 22.8%, 14.03%]
D:FilterOutStopKeep2000E:NbrInCluster4000	0.017	0.02	35[27.39%, 20.54%, 26.02%, 16.43%, 9.58%]
C:D:FilterOutStopKeep2000	0.017	0.019	35[30.88%, 22.05%, 22.05%, 5.88%, 19.11%]
D:FilterOutStopKeep2000E:NbrInCluster5000	0.018	0.018	36[27.27%, 13.63%, 30.3%, 18.18%, 10.6%]
A:Filter2ValuesE:NbrInCluster5000	0.018	0.037	37[26.22%, 18.03%, 19.67%, 21.31%, 14.75%]
A:Filter2ValuesE:NbrInCluster3000	0.018	0.034	37[27.27%, 23.63%, 25.45%, 12.72%, 10.9%]
A:Filter2ValuesE:NbrInCluster6000	0.02	0.045	40[27.63%, 26.31%, 23.68%, 11.84%, 10.52%]
A:Filter2Values_E:NbrInCluster50	0.02	0.042	41[29.03%, 32.25%, 14.51%, 19.35%, 4.83%]
A:Filter2ValuesE:NbrInCluster4000	0.021	0.04	42[35.29%, 26.47%, 14.7%, 16.17%, 7.35%]

Algorithm Name	Recall	Precision	Purchases
A:Filter2Values_E:NbrInCluster60	0.022	0.042	45[36.5%, 26.98%, 17.46%, 17.46%, 1.58%]
A:Filter2Values_B:Top1000	0.022	0.026	45[40.47%, 15.47%, 19.04%, 19.04%, 5.95%]
B:Freq5000_A:	0.023	0.03	47[24.0%, 25.0%, 18.0%, 30.0%, 3.0%]
A:2Values_B:Freq5000	0.023	0.029	47[35.16%, 25.27%, 21.97%, 10.98%, 6.59%]
A:Filter2ValuesB:Top5000	0.023	0.027	47[28.26%, 31.52%, 21.73%, 15.21%, 3.26%]
A:Filter2ValuesB:Freq5000	0.024	0.026	48[42.22%, 22.22%, 17.77%, 12.22%, 5.55%]
A:Filter2Values	0.025	0.029	51[34.0%, 33.0%, 15.0%, 13.0%, 5.0%]
A:2Values	0.025	0.028	51[34.4%, 26.88%, 18.27%, 11.82%, 8.6%]
B:Freq5000_A:2Values	0.026	0.027	52[37.77%, 23.33%, 16.66%, 12.22%, 10.0%]

Table A.1: Results from the short tests run for our own general algorithm on Company A's data set.

Algorithm Name	Recall	Precision	Purchases
D:_exact	0.004	0.021	9[80.0%, 20.0%, 0.0%, 0.0%, 0.0%]
topSellers	0.011	0.023	22[16.66%, 23.33%, 8.33%, 26.66%, 25.0%]
E:	0.019	0.019	38[35.29%, 25.49%, 21.56%, 7.84%, 9.8%]
A:	0.022	0.037	44[18.75%, 26.04%, 20.83%, 16.66%, 17.7%]
D:	0.029	0.036	59[29.41%, 21.17%, 18.82%, 20.0%, 10.58%]
B:	0.03	0.04	60[31.57%, 15.78%, 18.94%, 14.73%, 18.94%]
C:	0.047	0.062	94[25.97%, 26.62%, 14.28%, 20.12%, 12.98%]
B:_exact	0.085	0.13	169[32.1%, 21.4%, 20.29%, 14.76%, 11.43%]

Table A.2: Results from the short tests run for the baselines on Company A's data set.

A.1.2 Results Using Synergy

We also tested our algorithms by combining their results with Apptus' best algorithm in synergy to test its function as a backfiller. The results for that is shown in Table A.3 and the result for the same kind of test using the baselines together with Apptus' best individual algorithm is shown in Table A.4. Apptus' best individual algorithm had a recall of 0.033, precision value of 0.127 and 66 purchases for the short test.

Algorithm Name	Recall	Precision	Purchases
syn[E:NbrInCluster60_D:FilterOutStopKeep1000]	0.036	0.058	73[37.83%, 28.82%, 9.9%, 5.4%, 18.01%]
syn[A:Filter2Values_B:Top1000]	0.038	0.037	76[39.49%, 26.05%, 10.92%, 6.72%, 16.8%]
syn[A:Filter2ValuesE:NbrInCluster3000]	0.038	0.067	77[35.89%, 29.91%, 14.52%, 5.98%, 13.67%]
syn[C:_D:FilterOutStopKeep1000]	0.039	0.038	78[33.58%, 27.48%, 9.16%, 6.87%, 22.9%]
syn[A:2Values_B:Freq5000]	0.039	0.04	78[36.0%, 30.4%, 9.6%, 8.0%, 16.0%]
syn[A:Filter2ValuesE:NbrInCluster5000]	0.039	0.066	79[34.45%, 28.57%, 12.6%, 8.4%, 15.96%]
syn[C:D:FilterOutStopKeep2000]	0.04	0.035	80[36.8%, 28.8%, 13.6%, 4.8%, 16.0%]
syn[B:Freq5000_A:]	0.04	0.041	80[34.28%, 22.85%, 12.85%, 18.57%, 11.42%]
syn[A:Filter2ValuesE:NbrInCluster6000]	0.04	0.067	80[36.06%, 29.5%, 15.57%, 4.09%, 14.75%]
syn[E:NbrInCluster5000_D:FilterOutStopKeep2000]	0.04	0.05	81[30.46%, 25.78%, 16.4%, 8.59%, 18.75%]
syn[A:Filter2Values_E:NbrInCluster50]	0.041	0.076	82[33.6%, 27.86%, 14.75%, 9.01%, 14.75%]
syn[D:FilterOutStopKeep2000E:NbrInCluster5000]	0.041	0.037	82[32.33%, 27.06%, 16.54%, 7.51%, 16.54%]
syn[A:Filter2ValuesE:NbrInCluster4000]	0.041	0.072	83[34.35%, 25.95%, 13.74%, 10.68%, 15.26%]
syn[A:Filter2ValuesB:Freq5000]	0.041	0.041	83[37.85%, 28.57%, 12.85%, 5.0%, 15.71%]
syn[A:Filter2Values_E:NbrInCluster60]	0.041	0.074	83[37.7%, 24.59%, 13.11%, 13.11%, 11.47%]
syn[A:Filter2Values]	0.042	0.041	84[37.68%, 28.26%, 11.59%, 7.24%, 15.21%]
syn[A:Filter2ValuesB:Top5000]	0.042	0.04	85[35.29%, 29.41%, 14.7%, 8.08%, 12.5%]
syn[D:FilterOutStopKeep2000E:NbrInCluster4000]	0.043	0.04	86[32.86%, 25.17%, 13.28%, 12.58%, 16.08%]
syn[B:Freq5000_A:2Values]	0.043	0.04	87[32.83%, 30.59%, 10.44%, 8.2%, 17.91%]
syn[A:2Values]	0.043	0.043	87[36.11%, 27.77%, 11.11%, 10.41%, 14.58%]

Table A.3: Results from the short tests run for the best combinations of our own general algorithm on Company A's data set combined with Apptus' best individual algorithm using synergy.

Algorithm Name	Recall	Precision	Purchases
syn[AllBaselines]	0.033	0.127	66[36.0%, 30.0%, 11.0%, 5.0%, 18.0%]
syn[D:_exact]	0.035	0.099	71[41.58%, 27.72%, 13.86%, 4.95%, 11.88%]
syn[topSellers]	0.038	0.05	76[33.08%, 29.32%, 10.52%, 6.01%, 21.05%]
syn[E:]	0.041	0.048	83[33.33%, 29.36%, 12.69%, 8.73%, 15.87%]
syn[A:]	0.045	0.057	91[34.0%, 27.33%, 11.33%, 9.33%, 18.0%]
syn[B:]	0.048	0.059	96[34.48%, 25.51%, 11.72%, 11.03%, 17.24%]
syn[D:]	0.054	0.068	108[34.33%, 24.09%, 13.85%, 15.06%, 12.65%]
syn[C:]	0.065	0.08	131[33.99%, 30.04%, 10.34%, 13.79%, 11.82%]
syn[B:_exact]	0.096	0.135	191[33.89%, 22.37%, 15.25%, 16.27%, 12.2%]

Table A.4: Results from the short tests run for the baselines on Company A's data set combined with Apptus' best individual algorithm using synergy.

A.2 Results for Company B

In this section the results from the short test on Company B's data set will be presented. The presentation will follow the same pattern as for Company A.

A.2.1 Individual Algorithm Results

The results for the shorts tests on our own algorithms are presented in Table A.4. Which connection algorithm was used can be inferred from the ending of the algorithm name, i.e., those ending with MAH1 use connection algorithm 1, MAH2 means connection algorithm 2 was used and so on. The reason why MAH was used in front of the numbers is that we had to name them something and we ended up using the Swedish acronym for Malmö University since the project has ties to it.

The combinations with connection algorithm 2 outperformed both those with connection algorithm 1 and connection algorithm 3 when looking at individual results so we decided to use that one. The results for the short tests on the baselines are shown in Table A.5.

Algorithm Name	Recall	Precision	Purchases
G:Freq5000_MAH2	0.077	0.115	398[28.52%, 27.99%, 19.27%, 13.97%, 10.23%]
G:Freq5000_H:_MAH2	0.104	0.112	537[25.02%, 21.5%, 20.41%, 19.17%, 13.88%]
H:_MAH2	0.104	0.113	538[24.81%, 20.84%, 20.71%, 17.73%, 15.88%]
H:_G:Freq5000_MAH2	0.105	0.112	542[24.8%, 22.37%, 19.66%, 19.47%, 13.68%]
G:Freq5000H:_MAH2	0.105	0.113	543[24.57%, 21.13%, 20.54%, 17.78%, 15.97%]
J:_G:Freq5000_MAH2	0.107	0.104	553[30.62%, 22.26%, 15.48%, 16.22%, 15.38%]
G:Freq5000J:_MAH2	0.107	0.105	553[30.22%, 22.83%, 15.69%, 15.59%, 15.64%]
G:Freq5000_J:_MAH2	0.107	0.103	555[30.69%, 22.38%, 15.67%, 15.07%, 16.16%]
J:_MAH2	0.107	0.105	555[30.24%, 22.87%, 15.7%, 15.56%, 15.6%]
H:_J:_MAH2	0.117	0.096	605[27.15%, 23.19%, 17.19%, 16.28%, 16.17%]
H:J:_MAH1	0.119	0.07	616[25.86%, 24.2%, 17.6%, 15.08%, 17.24%]
F:H:_MAH2	0.122	0.096	633[29.14%, 21.77%, 17.72%, 16.78%, 14.57%]
I:_MAH2	0.123	0.099	637[28.14%, 19.53%, 15.36%, 18.5%, 18.45%]
G:Freq5000F:_MAH2	0.123	0.104	638[30.75%, 22.31%, 19.36%, 14.03%, 13.52%]
F:_G:Freq5000_MAH2	0.125	0.104	648[29.49%, 24.31%, 18.66%, 14.2%, 13.32%]
H:_F:_MAH2	0.127	0.1	660[27.56%, 21.28%, 16.7%, 16.97%, 17.46%]
G:Freq5000_F:_MAH2	0.127	0.107	660[31.58%, 21.97%, 18.78%, 14.59%, 13.05%]
F:J:_MAH1	0.128	0.072	662[31.42%, 24.01%, 19.06%, 13.77%, 11.72%]
F:_H:_MAH2	0.128	0.099	662[29.33%, 20.85%, 16.73%, 17.23%, 15.83%]
F:_MAH2	0.129	0.109	669[30.55%, 22.51%, 19.59%, 14.81%, 12.52%]
H:J:_MAH2	0.134	0.099	693[26.55%, 21.99%, 17.27%, 15.79%, 18.38%]
J:_H:_MAH2	0.135	0.099	698[26.77%, 22.01%, 17.15%, 15.53%, 18.51%]
J:_F:_MAH3	0.14	0.08	726[29.62%, 23.2%, 20.68%, 13.72%, 12.76%]
A:I:_MAH2	0.143	0.093	741[29.65%, 19.53%, 18.82%, 14.86%, 17.12%]
F:J:_MAH2	0.146	0.1	758[29.16%, 23.92%, 15.03%, 15.08%, 16.78%]
J:_F:_MAH2	0.147	0.099	762[29.96%, 23.1%, 15.69%, 14.87%, 16.36%]
F:_J:_MAH2	0.148	0.1	766[29.45%, 22.34%, 15.62%, 14.89%, 17.69%]
A:_G:Freq5000_MAH3	0.156	0.092	809[33.37%, 24.55%, 15.37%, 16.2%, 10.48%]
G:Freq5000A:_MAH1	0.16	0.085	827[34.66%, 24.0%, 14.94%, 14.71%, 11.67%]
G:Freq5000A:_MAH3	0.165	0.092	854[34.58%, 24.15%, 14.7%, 14.92%, 11.62%]
A:_I:_MAH2	0.171	0.1	885[25.9%, 20.12%, 19.78%, 17.61%, 16.58%]
I:_A:_MAH2	0.172	0.099	888[26.8%, 17.87%, 21.54%, 18.21%, 15.55%]
A:_G:Freq5000_MAH2	0.174	0.119	898[28.44%, 20.19%, 19.69%, 15.51%, 16.15%]
A:_J:_MAH2	0.174	0.108	900[29.57%, 19.96%, 19.61%, 16.75%, 14.09%]
J:_A:_MAH2	0.174	0.108	901[29.54%, 19.94%, 19.59%, 16.83%, 14.08%]
A:J:_MAH2	0.175	0.109	905[28.14%, 19.01%, 20.74%, 17.18%, 14.91%]
G:Freq5000A:_MAH2	0.176	0.117	909[29.49%, 19.99%, 19.13%, 15.57%, 15.8%]
G:Freq5000_A:_MAH2	0.176	0.119	910[29.74%, 19.76%, 19.45%, 15.92%, 15.11%]
A:_MAH2	0.177	0.119	915[29.56%, 19.63%, 19.63%, 15.8%, 15.36%]
A:_H:_MAH2	0.177	0.113	918[31.16%, 18.11%, 19.74%, 16.28%, 14.69%]
F:A:_MAH2	0.179	0.114	924[29.93%, 20.27%, 19.33%, 15.27%, 15.17%]
H:_A:_MAH2	0.182	0.11	939[31.02%, 16.65%, 19.07%, 17.54%, 15.7%]
F:_A:_MAH2	0.182	0.108	944[31.0%, 20.59%, 18.24%, 16.06%, 14.1%]
A:H:_MAH2	0.185	0.113	957[31.08%, 17.21%, 18.86%, 17.21%, 15.61%]
A:_F:_MAH2	0.187	0.112	968[30.19%, 21.34%, 17.63%, 15.46%, 15.36%]

Table A.4: Results from the short tests run for our own general algorithm on Company B's data set.

Algorithm Name	Recall	Precision	Purchases
G:	0.0	0.002	1[0.0%, 100.0%, 0.0%, 0.0%, 0.0%]
Topsellers	0.07	0.114	365[28.09%, 23.67%, 18.4%, 14.96%, 14.87%]
H:	0.081	0.109	421[25.42%, 22.78%, 18.69%, 17.7%, 15.39%]
A:	0.084	0.111	437[30.88%, 21.03%, 14.52%, 16.17%, 17.36%]
I:	0.12	0.073	620[37.14%, 20.7%, 16.58%, 14.81%, 10.75%]
J:	0.12	0.092	623[29.93%, 23.17%, 20.25%, 13.59%, 13.04%]
F:	0.121	0.091	626[26.51%, 23.51%, 17.08%, 17.63%, 15.25%]

Table A.5: Results from the short tests run for the baselines on Company B's data set.

Algorithm Name	Recall	Precision	Purchases
syn[G:Freq5000_A:_MAH2]	0.185	0.104	938[29.1%, 20.42%, 18.95%, 15.75%, 15.75%]
syn[A:_G:Freq5000_MAH2]	0.181	0.117	938[28.2%, 19.91%, 19.91%, 15.56%, 16.41%]
syn[F:_MAH2]	0.184	0.115	951[25.89%, 25.71%, 18.88%, 16.08%, 13.42%]
syn[I:_MAH2]	0.185	0.104	958[26.52%, 24.95%, 16.43%, 16.53%, 15.55%]
syn[G:Freq5000A:_MAH2]	0.188	0.118	972[28.28%, 20.83%, 19.42%, 14.31%, 17.13%]
syn[J:_F:_MAH2]	0.188	0.105	974[27.05%, 21.16%, 18.37%, 18.87%, 14.52%]
syn[F:A:_MAH2]	0.19	0.114	985[29.0%, 20.49%, 20.4%, 14.22%, 15.87%]
syn[F:J:_MAH2]	0.191	0.105	986[26.25%, 23.52%, 20.25%, 16.24%, 13.71%]
syn[G:Freq5000_MAH2]	0.191	0.118	990[22.74%, 26.19%, 15.2%, 23.35%, 12.49%]
syn[J:_A:_MAH2]	0.198	0.108	1025[28.18%, 20.18%, 19.25%, 16.04%, 16.33%]
syn[J:_MAH2]	0.199	0.113	1028[24.67%, 27.29%, 16.61%, 14.39%, 17.02%]
syn[A:_MAH2]	0.2	0.118	1036[28.98%, 19.42%, 19.33%, 17.22%, 15.02%]
syn[H:_MAH2]	0.201	0.104	1041[26.44%, 26.15%, 18.44%, 16.38%, 12.56%]
syn[H:J:_MAH2]	0.201	0.1	1041[28.46%, 24.05%, 18.15%, 16.51%, 12.82%]
syn[A:_J:_MAH2]	0.202	0.109	1044[28.32%, 19.61%, 19.07%, 17.86%, 15.1%]
syn[J:_F:_MAH3]	0.203	0.098	1049[26.78%, 19.92%, 18.74%, 19.55%, 14.99%]
syn[A:_F:_MAH2]	0.204	0.109	1056[29.22%, 21.07%, 18.16%, 15.74%, 15.79%]
syn[G:Freq5000A:_MAH3]	0.205	0.103	1058[27.96%, 24.54%, 19.81%, 13.83%, 13.83%]
syn[A:_H:_MAH2]	0.205	0.113	1061[30.27%, 18.33%, 18.56%, 16.25%, 16.57%]
syn[H:J:_MAH1]	0.208	0.092	1076[29.6%, 20.88%, 17.65%, 16.17%, 15.67%]
syn[I:_A:_MAH2]	0.209	0.102	1082[27.28%, 22.5%, 17.51%, 15.85%, 16.83%]
syn[A:H:_MAH2]	0.209	0.11	1083[30.71%, 19.8%, 16.09%, 18.0%, 15.38%]
syn[H:_A:_MAH2]	0.21	0.108	1084[29.29%, 21.04%, 17.67%, 17.13%, 14.84%]
syn[F:_A:_MAH2]	0.21	0.106	1084[29.84%, 20.08%, 17.6%, 17.7%, 14.76%]
syn[F:J:_MAH1]	0.21	0.095	1084[28.66%, 20.13%, 17.31%, 14.65%, 19.23%]
syn[A:_G:Freq5000_MAH3]	0.21	0.101	1086[29.61%, 23.15%, 17.93%, 14.83%, 14.47%]
syn[A:J:_MAH2]	0.211	0.111	1089[24.7%, 24.65%, 17.02%, 18.82%, 14.79%]
syn[G:Freq5000A:_MAH1]	0.213	0.1	1101[27.56%, 21.4%, 19.48%, 16.35%, 15.19%]

Table A.6: Results from the short tests run for the best combinations of our own general algorithm on Company B's data set combined with Apptus' best individual algorithm using synergy.

A.2.2 Results Using Synergy

The approach using synergy was used for Company B's data set as well and the results of the short tests for our own algorithms and the baselines are presented in Tables A.6 and A.7 respectively. Apptus' best individual algorithm had a recall of 0.21, precision value of 0.096 and 1087 purchases for the short test.

Algorithm Name	Recall	Precision	Purchases
syn[H:]	0.171	0.118	883[22.91%, 25.81%, 16.64%, 16.34%, 18.28%]
syn[J:]	0.207	0.105	1072[27.23%, 18.59%, 15.88%, 18.1%, 20.18%]
syn[G:]	0.21	0.096	1088[29.6%, 21.46%, 18.06%, 15.98%, 14.88%]
syn[F:]	0.211	0.096	1092[28.24%, 20.65%, 17.33%, 16.06%, 17.7%]
syn[I:]	0.214	0.097	1109[27.45%, 24.64%, 19.07%, 14.57%, 14.25%]
syn[A:]	0.216	0.1	1118[27.98%, 22.59%, 20.4%, 16.03%, 12.97%]

Table A.7: Results from the short tests run for the baselines on Company B's data set combined with Apptus' best individual algorithm using synergy.

Appendix B
How To Configure the Conditions File

To run the system, a file with configurations for how the clustering should work is needed. In this appendix, how that file should be formatted and the different options available are described.

B.1 Formatting

The configuration file should have the following format:

```
Levels
    NameOfClusteringClass
        Transformation
            AttributeName NameOfTransform Weight Arguments
            AttributeName NameOfTransform Weight Arguments
        END
        Filters
            AND
                AttributeName NameOfFilter Arguments
            END
        END
    END
END
```

Several blocks enclosing NameOfClusteringClass to its END tag can be added to get hierarchical clustering. The Transformation and Filters tags must be defined but they may be empty, i.e. write Filter on one line and END on the next one. The filters can also specifiy AND and OR conditions using the tags AND and OR respectively. If filters are defined, one should use enclosing AND/OR tags for the program to actually read the filters.For

example, this could be used for putting in the condition that you both want the product to have 2 values of attribute A and 1 value of attribute B. At the end of this appendix there will be an example of a detailed configurations file.

B.2 Transformation Classes

There are two main classes for handling data, one for handling numerical data and one for handling categorical (or n-valued) data. The one for handling categorical data has been extended into several transformation classes that uses content filters of one type or another. It is easy to extend them to create new classes if they are needed. The names of the transformation classes and their uses are described below.

B.2.1 TransformNValued

The TransfomNValued transform is the base class for the categorical value transforms. It is optimal to use if you don't have any requirements on the values that are included and it does not take any arguments. It can handle several values for an attribute.

B.2.2 TransformNValuedFilterPrefix

The TransformNValuedFilterPrefix transform inherits TransformNValued and has the same functionality but places restrictions on the values accepted by defining accepted prefix of the values. For example, if one has q as the input argument to the transform it will only keep the values that starts with q. The default argument for this transform if none is sent in is the empty string. It does not allow for several prefixes as input arguments but the class could easily be extended or changed to allow for that.

B.2.3 TransformNValuedFilterContains

The TransformNValuedFilterContains transform inherits TransformNValuedFilterPrefix and has the same functionality except it will look for the input argument somewhere in the value for the attribute instead of just at the beginning. Same restrictions and default value apply as for the TransformNValuedFilterPrefix class.

B.2.4 TransformNValuedFilterNTopSeller

The TransformNValuedFilterNTopSeller transform inherits TransformNValued. It works by querying eSales which n values for the attribute have been most common among the products that have been sold so far. Since this is market specific and also dependent on the time as new user data becomes available as time goes, this transform means the clustering needs to be redone every once in a while. There is a boolean variable in the TopLevel class called marketDependentClustering that regulates whether or not this reclustering should be done. The transform takes one input argument – the number n. The default value for n if no argument is given is 5000.

B.2.5 TransformNValuedFilterNFrequent

The TransformNValuedFilterNFrequent transform is similar to TransformNValuedFilterNTopSeller but instead of querying eSales for the top sellers it uses the product catalogue information and keeps the n most frequent values for the attribute from there. It takes the same input argument as TransformNValuedFilterNTopSeller and has the same default value.

B.2.6 TransformNValuedFilterOutStopKeepNFrequent

The TransformNValuedFilterOutStopKeepNFrequent transform (yes, I know the names are getting ridiculously long) is similar to TransformNValuedFilterNFrequent but is intended for attributes that contain so called stop words, i.e. words that are common but don't really give much information like "and", "but" and so on. This transform has two input arguments, the number of frequent values to keep (n) and the number of most common values that we think won't give us anything. They should be given in that order. The default values for the parameters are 5000 for n and 20 for the stop words.

B.2.7 TransformNValuedFilterNFrequentTokenize

The TransformNValuedFilterNFrequentTokenize transform inherits TransformNValuedFilterNFrequent and is very similar to it. The only difference is that it tokenises all values and then sorts the tokens instead of the values, with the sorting based on frequency in the product catalogue. It takes the same input argument and has the same default value as its super class.

B.2.8 TransformNumerical

The TransformNumerical transform is intended to be used for attributes with numerical values. It requires two arguments – the bucket definition class used to create the buckets the values are placed in and how many items one would like to be in each bucket. If these arguments are not supplied, an IOException will be thrown.

There are currently two bucket definition classes available, MedianSpacedBuckets and EvenlySpacedBuckets. The EvenlySpacedBucket one does pretty much what it sounds like, it calculates the difference between the highest and lowest value and then defines the difference between each bucket boundary as the global difference divided by the number of buckets. The number of buckets is calculated from the number of items divided by the specified number of items per bucket. The MedianSpacedBuckets calculates its bucket boundaries by sorting the values and then placing the specified number of items per bucket in each bucket until it runs out of values. If a value and the one used as a boundary value are the same, it jumps over that value for the next bucket. The TransformNumerical transform will add a double element at the end of the array as an indicator that this attribute has no values for this item.

B.3 Filter Classes

There is only one filter class at present to specify conditions on the number of values an item should have for an attribute to be included in the clustering, the NValuedFilter class. It takes one input argument – the number of values an item should for an attribute. The default value if no argument is given is 1. If one chooses to use the OR or AND keywords there are corresponding filter classes that handle that but the user does not need to specify the names of those.

B.4 Clustering Classes

There are currently two clustering classes, SimpleClusters and DoubleBucketBase. The SimpleClusters class uses KMeans++ to cluster the items. The items that are clustered in the initial clusters are those that passed the filter specified by the FILTER keyword. The rest of the items are then put in the cluster it has the closest distance to (without changing the centroid) or creates its own cluster if none of the values match.

The DoubleBucketBase class is used when clustering (or bucketing) attributes with numerical values. It assumes that the transform class used is either TransformNumerical or a future extension of that class.

B.5 A Detailed Configuration File Example

Below is an example of a detailed configuration file. On the first level of the clustering SimpleClusters is used to cluster Attribute_A and Attribute_C. For Attribute_C, only the values of the 5000 most frequent values are used after the 20 most frequent ones are removed. Only the items that have at least one of these values and at least one value for Attribute_A will be clustered in the initial clustering and the rest are added as described in B.4. Then, for each cluster, DoubleBucketBase will be used for clustering on Attribute_B. MedianSpacedBuckets will be used for creating the bucket definition and the number of items wished for each cluster is 50. There are no filters specified for this level.

```
Levels
   SimpleClusters
      Transformation
         Attribute_A TransformNValued 0.5
         Attribute_C TransformNValuedFilterOutStopKeepNFrequent 0.5 5000 20
      END
      Filters
         AND
            Attribute_A NValuedFilter 1
            Attribute_C NValuedFilter 1
         END
      END
```

```
    END
    DoubleBucketBase
        Transformation
            Attribute_B TransformNumerical 1 MedianSpacedBuckets 50
        END
        Filters
        END
    END
END
```