Learning Hybrid Poisson Aspect Model for Personalized Shopping Recommendation

Advisor

Professor Chun-Nan Hsu

Student

Hao-Hsiang Chung

A Thesis Submitted to

Department of Computer Science

and Information Engineering

National Taiwan University

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Computer Science

June, 2002

Taipei, Taiwan, Republic of China

Acknowledgements

I would like to express my gratitude to my advisor, Prof. Chun-Nan Hsu, for his patient guidance and technical advice during the preparation of this thesis. I want to thank my committees, Prof. Shun-Chin Hsu and Prof. Chih-Jen Lin, for their helpful comments. I also want to thank my senior Han-Shen Huang whose instruction and recommendation directed me when I was in trouble. Finally, I would like to thank my parents for their patience and support as I was working on this thesis.

Abstract

Recommendations that really match customers' needs can boost sales. Researchers have proposed and evaluated many approaches for generating recommendations. In this thesis, we proposed a model-based collaborative approach, called *Hybrid Poisson Aspect Model* (*HyPAM*). HyPAM is a hybrid system combining two probabilistic models, *cluster* and *aspect*, which model the relationship between customer clusters and product types. Given a new customer and his/her shopping record, HyPAM can estimate his/her degree of preference of each product item accurately. We use the EM algorithm to learn the parameters of HyPAM from customers' shopping records. To evaluate our approach, we apply HyPAM and two well-known recommender systems, *GroupLens* and *IBM*, to a shopping-record data set provided by a local supermarket. This data set contains 119,578 transactions of 32,266 distinguishable customers in a four-month period. We adopted two metrics, *rank score* and *lift index*, with four protocols, *Given 2*, *Given 5*, *Given 10*, and *All but one.* Under these evaluation metrics, experimental results show that HyPAM performs much better compared to the other two recommendation approaches for the given data set.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

People always have less time but more decisions to make. “What movie should I see?” “What book should I read?” “What product should I buy?” There are far too many choices and we do not have time to explore them all. A recommender system can help people make such complex decisions. Recommenders suggest the user items that he/she may like. A good recommendation may save our time and money and boost sales for merchants.

A supermarket is a place where we have to make plenty of decisions. Because there are a lot of choices, we need a lot of help too. In this thesis, we develop a probability model called *Hybrid Poisson Aspect Model* (HyPAM) to construct a recommender system for personalized shopping recommendation. We use a set of four-month supermarket shopping data to train our model, and have a better performance and less prediction time in comparison with other recommender systems.

1.2 Related Works

One of the earliest and most successful recommender systems is *collaborative filtering.* The GroupLens research system [7] provided a pseudonymous collaborative filtering solution for Usenet news and movies. Ringo [8] made personalized recommendations for music albums and artists. Other model-based collaborative filtering algorithms are also exploited, such as Bayesian networks and clustering. Breese et al. [2] proposed to use a specific Bayesian network model based on a training set with a decision tree at each node and the edges in the model representing user information. They also proposed clustering techniques work by identifying groups of users who appear to have similar preference. Finally, Breese et al. reported experiments to compare all these work. Agrawal et al. [1] developed well-known association rule mining algorithms to analysis shopping records in supermarkets. There are recommender systems designed based on association rule mining. Lawrence et al. [5] from IBM proposed a recommender system for supermarket products. Their system which is a hybrid system of content-based and collaborative filtering.

1.3 Organization

The remainder of this thesis is organized as follows: Chapter 2 reviews some recommender systems we have studied. Chapter 3 discusses the detail of our proposed model, HyPAM. Chapter 4 presents the experimental evaluations. Finally, we give a conclusion in Chapter 5.

Chapter 2

Review of Well-Known Recommender Systems

Recommender systems apply data analysis techniques to the problem of helping customers find which products they would like to purchase. The underlying techniques used in today's recommender systems fall into two filtering methods: *content-based filtering* and *collaborative filtering* methods. In content-based recommendation one tries to recommend items similar to those a given user has liked in the past, whereas in collaborative recommendation one identifies users whose preferences are similar to those of the given user and recommends items they have liked.

In this chapter, we discuss several recommender systems that will be compared with our algorithm, HyPAM, in our experimental studies.

2.1 Association Rules

One of the most commonly used data mining techniques for recommender systems is to find association rules [1] within the transactional data set. These techniques are concerned with discovering association between two distinct sets of products. Formally, let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a collection of m products. Let $\mathcal{D}$ be a set of transactions, where each transaction T is a set of products such that $T \subseteq \mathcal{I}$. An *association rule* between two sets of products X and Y, such that $X, Y \subseteq \mathcal{I}$ and $X \cap Y = \emptyset$, states that the preference of products in the set X in the transaction T indicates a strong likelihood that products from the set Y are also present in T.

There are two measurements to evaluate the association rules, ***support*** and ***confidence***. The support s, of a rule measures the occurrence frequency of the pattern in the rule while the confidence c is the measure of the strength of implication. For a rule $X \Rightarrow Y$, the support is measured by the fraction of transactions that contains both X and Y. More formally,

$$s = \frac{\text{number of the transactions containing } X \cup Y}{\text{number of transactions}}.$$

In other words, support indicates that $s \times 100\%$ of transactions contain $X \cup Y$. For a rule $X \Rightarrow Y$, the confidence c states that $c \times 100\%$ of transactions that contain X also contain Y. More formally,

$$c = \frac{\text{number of the transactions containing } X \cup Y}{\text{number of transactions containing X}}.$$

With association rules, it is common to find rules having support and confidence higher than a user-defined minimum.

Association rules can be applied to develop recommender systems in the following way. For each customer, one creates a set of products that he/she has purchased in the past. Then one uses an association rule discovery algorithm to find all the rules that satisfy given minimum support and minimum confidence constraints. Now, for each customer u one can generate his/her recommendation list, or *top-N* recommended products as follows. First, one finds all the rules that are *supported* by the customer (i.e. the customer has purchased all the products that are in the left-hand-side of the rule). Let P_u be the set of products that are predicted by all these rules (i.e. in the right-hand-side of these rules) and have not yet be purchased by customer u. Next, one sorts these products according to the confidence of the rules that were used to predicted them, so that the higher the confidence is, the higher the ranked position is. Note that if a particular product is given from multiple rules, one can use the rule that has the highest confidence. Finally, one selects the first N highest ranked products as the recommendation list of customer u.

2.2 GroupLens

GroupLens [7] is a collaborative filtering system applied to Usenet news. The aim of GroupLens is to help people find articles they will like in the huge stream of available articles. News reader clients display predicted scores and make it easy for users to rate articles after they read them. The prediction algorithm is based on a simple intuition: predictions for a user should be based on the similarity between the interest profile of that user and those of other users. To implement this heuristics, the first step is to compute similarities between user profiles. By assigning the similarities (or weights) with others to the target user, the predicted score can be computed as the

article #	Ken	Lee	Meg	Nan
1	1	4	2	2
2	5	2	4	4
3			3	
4	2	5		5
5	4	1		1

Figure 2.1: Rating Matrix

weighted sum of the ratings of other persons for current article.

The user database therefore consists of a set of ratings $r_{i,j}$ corresponding to the rating for user i on article j. Fig. 2.1 shows an example of these raings data. Let I_i be the set of articles on which user i has rated. The mean rating for user i is

$$\bar{r}_i = \frac{1}{|I_i|} \sum_{j \in I_i} r_{i,j}.$$

The predicted rating of the active user (indicated with a subscript a) for article j, $p_{a,j}$, is a weighted sum of the ratings of the other users:

$$p_{a,j} = \bar{r}_a + \kappa \sum_{i=1}^{n} w(a,i)(r_{i,j} - \bar{r}_i)$$

where n is the number of users in the database, $w(a,i)$ is the similarity between user i and the active user, and κ is a normalizing factor such that the absolute values of the similarities sum to unity. The similarity $w(a,j)$ adopted in GroupLens is the *Pearson correlation coefficient:*

$$w(a,j) = \frac{\sum_j (r_{a,j} - \bar{r}_a)(r_{i,j} - \bar{r}_i)}{\sqrt{\sum_j (r_{a,j} - \bar{r}_a))^2 \sum_j (r_{i,j} - \bar{r}_i))^2}}.$$

Other type of similarity is possible, like vector similarity:

$$w(a,i) = \sum_j \frac{r_{a,j}}{\sqrt{\sum_{k \in I_a} r_{a,k}^2}} \frac{r_{i,j}}{\sqrt{\sum_{k \in I_i} r_{i,k}^2}}$$

2.3 Bayesian Clustering

Breese et al. [2] identify two major classes of collaborative filtering algorithms. *Memory-based* algorithms operate over the entire user database to make predictions. In *Model-based* collaborative filtering, in contrast, uses the user database to estimate or learn a model, which is then used for predictions. While GourpLens is a type of memory-based collaborative filtering, the Bayesian clustering that will be discussed in this section is a type of model-based collaborative filtering.

The clustering model treats collaborative filtering as a classification problem [2] and works by clustering similar users in the same class and estimating the probability that a particular user is in a particular class C, and from there computes the conditional probability of ratings. The Bayesian clustering holds a naive Bayes assumption: given the class, the preferences (ratings) regarding the various items (articles) are independent. Thus the probability of class and ratings is given by

$$Pr(C = c, r_1, \ldots, r_n) = P(C = c) \prod_{i=1}^{n} P(r_i | C = c).$$

The prediction task can be viewed as calculating the expected value of a rating, given what we know about the active user. If we assume the ratings are integer valued range from 0 to m we have

$$p_{a,j} = E(r_{a,j}) = \sum_{i=0}^{m} P(r_{a,j} = i | r_{a,k}, k \in I_a) i,$$

where $P(r_{a,j} = i | r_{a,k}, k \in I_a)$ can be written as

$$\begin{aligned} P(r_{a,j} = i | r_{a,k}, k \in I_a) &= \sum_{c} P(r_{a,j} = i | C = c) P(C = c | r_{a,k}, k \in I_a) \\ &= \sum_{c} P(r_{a,j} = i | C = c) \frac{P(r_{a,k}, k \in I_a | C = c) P(C = c)}{\sum_c P(r_{a,k}, k \in I_a | C = c) P(C = c)}. \end{aligned}$$

2.4 IBM Method

Lawrence et al. [5] proposed a recommender system for supermarket products. The recommender system has been implemented as part of the "SmartPad" remote shopping system developed by IBM and Safeway Stores, a major supermarket retailer in the U.K.

This method uses content-based filtering at its core, with the idea from collaborative filtering to refine the content model and to make recommendation dependent on shared interests within customer clusters. Each customer and product is represented by a feature vector. Products to be recommended can then be determined by computing a measure of distance between vectors representing personal preferences and vectors representing products.

The product taxonomy used by "Safeway Stores" has a three-level hierarchy. Products are divided into 99 product classes. Each product class is subdivided into fewer than 100 subclasses, generating a total of 2032 product subclasses. The customer spending at each level of the hierarchy is available.

The major task of this recommender system is to construct customer vector model and product vector model. In the following, we discussed the procedures to construct customer and product models.

Customer Model

The absolute spending for customer m is represented as

$$C^{(m)} = [C_{m1}, \dots C_{ms} \dots, C_{mS}]^T, m = 1, \dots, M,$$

where C_{ms} denotes the absolute spending of customer m across all products contained in subclass s, M is the total number of customers, and S is the number of product subclasses. Two separate normalizations are applied to this result to obtain the final customer vector.

1. *Self-Normalizing*
$$\hat{C}_{ms} = \frac{C_{ms}}{\sum_{s'=1,\ldots,S} C_{ms'}}$$
2. *Normalizing with other customers*
$$\hat{\hat{C}}_{ms} = \frac{\hat{C}_{ms}}{\frac{1}{M}\sum_{m'=1,\ldots,M} \hat{C}_{m's}}$$

Product Model

Each product $n = 1, \ldots, N$ is represented by a S-dimensional vector $\mathbf{P}^{(n)}$, and hence has the same dimensionality with the customer vectors. The individual entries $\mathbf{P}^{(n)}_s$, $s = 1, \ldots, S$, reflect the "affinity" the product has to the subclass s. By using the association rules on both class and subclass levels (e.g., bacon and eggs are associated classes; bacon/lowfat and eggs/medium are associated subclasses), the product vector can be constructed by the following rules:

$$P^{(n)}_s = \begin{cases} 1.0 & \text{if } s = S(n) & \text{(within the same subclass)} \\ 1.0 & \text{if } S(n) \Rightarrow s & \text{(within associated subclass)} \\ 0.5 & \text{if } C(s) = C(n) & \text{(subclass within the same class)} \\ 0.25 & \text{if } C(n) \Rightarrow C(s) & \text{(within subclass of associated class)} \\ 0 & \text{otherwise} \end{cases},$$

where

$$\begin{aligned} S(n) &\equiv \textit{the product subclass number for the product } n \\ C(s) &\equiv \textit{the product class for the product subclass } s \\ C(n) &\equiv C(S(n)). \end{aligned}$$

Matching Algorithm After the customer and product vector models have been constructed, the final step in the recommendation process is to score each candidate product for a specific customer and select the best matches. The score σ_{mn} between customer m and product n is computed using a cosine coefficient between the corresponding vectors, $\mathbf{C}^{(m)}$ and $\mathbf{P}^{(n)}$:

$$\sigma_{mn} = \rho_n \frac{\mathbf{C}^{(m)} \cdot \mathbf{P}^{(n)}}{||\mathbf{C}^{(m)}|| ||\mathbf{P}^{(n)}||},$$

where ρ_n is an identical characteristic factor for each product.

Chapter 3

Hybrid Poisson Aspect Model

We have discussed four well-known recommender systems in Chapter 2. In this chapter, we propose an alternative method based on a probability model. We call it "Hybrid Poisson Aspect Model (HyPAM)," which is a combination of two probability models, cluster and aspect models. In Section 3.1, we give the descriptions of the two models and how they are combined together. In Section 3.2, we show how to construct HyPAM for supermarket shopping data using the EM algorithm. In Section 3.3, we show how this model works.

3.1 Model Specification

The HyPAM uses a cluster model to classify customers into some predefined clusters and an aspect model to encode the relationships between clusters and products.

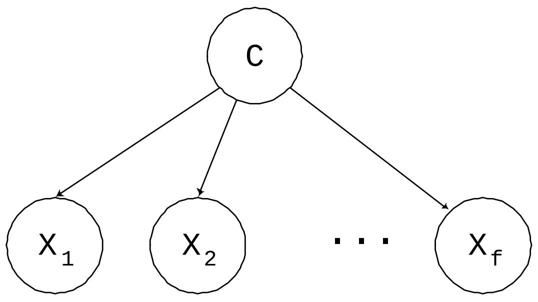


Figure 3.1: Cluster Model

3.1.1 Cluster Model

Assuming that each input data can be represented by some features. It is plausible to apply a Bayesian classifier to the data where the probability distributions of the features are conditionally independent given their membership in an unobserved class variable C with some relatively small number of values. This model is also known as a *naive Bayes classifier.* Fig. 3.1 gives a graphical depiction.

Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ be the data set. Each data instance $\mathbf{x}_m$ is a f-dimension vector $\mathbf{x}_m = (x_{m1}, x_{m2}, \ldots, x_{mf})$, for $m = 1, \ldots, M$. $C \in \{c_1, c_2, \ldots, c_G\}$ is an unobserved class variable. Given the class label, the probability distributions of the feature values are conditionally independent. The probability of a data vector given its class is

$$P(\mathbf{x}_m|C = c_g) = \prod_{i=1}^{f} P(x_{mi}|C = c_g)$$

and the joint probability of class and features is

$$P(C = c_g, x_1, ..., x_f) = P(C = c_g) \prod_{i=1}^{f} P(x_i|C = c_g).$$

The model parameters, $P(C = c_g)$ and $P(x_i|C = c_g)$, are estimated from the training data. Since the value of the class variables are not available in the database, this is a problem of learning parameters for models with hidden variables. We must employ methods that can deal with hidden variables, like EM algorithm [3]. By summing all the possible values of the class variable C, the complete data probability is given by

$$\begin{aligned} P(\mathbf{X}) &= \prod_{m=1}^{M} P(\mathbf{x}_m), \quad \text{where} \\ P(\mathbf{x}_m) &= \sum_{g=1}^{G} P(c_g) P(\mathbf{x}_m|c_g) \\ &= \sum_{g=1}^{G} P(c_g) \prod_{i=1}^{f} P(x_{mi}|c_g). \end{aligned}$$

After the learning process, we can get the posterior of an input data vector

$$P(c_g|\mathbf{x}_m) = \frac{P(c_g)P(\mathbf{x}_m|c_g)}{\sum_{g'=1}^{G} P(c_{g'})P(\mathbf{x}_m|c_{g'})}.$$

This gives a “soft” classification, since we do not assign exactly one cluster to each input data vector but a probability associated with each cluster instead.

3.1.2 Aspect Model

Hofmann [4] proposed an *aspect model* — a latent class statistical mixture model — for co-occurrence data with a set of latent variables. Consider two sets of objects, $\mathcal{X} = \{x_1, \ldots, x_I\}$ and $\mathcal{Y} = \{y_1, ..., y_J\}$, and their co-occurrence data set, $\mathbf{S} = (x^n, y^n)_{1 \leq n \leq N}$. Each pair (x^n, y^n) is associated with a latent variable A^n over some finite set $\mathcal{A} = \{a_1, ..., a_K\}$. Observations that share the same class are simply referred to as an aspect. The random variables X^n and Y^n are conditionally independent given the respective latent class A^n. Fig. 3.2 shows the graphical representation of this model. The

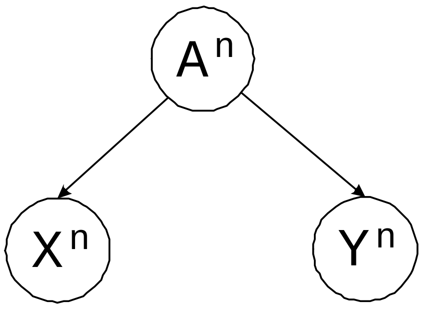


Figure 3.2: Aspect Model

complete data probability is given by

$$\begin{aligned} P(\mathbf{S}, \mathbf{a}) &= \prod_{n=1}^{N} P(x^n, y^n, a^n), \quad \text{where} \\ P(x^n, y^n, a^n) &= P(a^n) P(x^n|a^n) P(y^n|a^n). \end{aligned}$$

By summing over all the possible values of the latent variables and grouping identical co-occurrence data together, we obtain the mixture probability of the data

$$\begin{aligned} P(\mathbf{S}) &= \prod_{x \in \mathcal{X}} \prod_{y \in \mathcal{Y}} P(x, y)^{n(x,y)} \quad \text{with} \\ P(x, y) &= \sum_{a \in \mathcal{A}} P(a) P(x|a) P(y|a) \\ n(x, y) &= |\{(x^n, y^n) : x^n = x \wedge y^n = y\}|. \end{aligned}$$

3.1.3 Hybrid Model

The cluster model partitions the input data represented by some features into G groups while the aspect model partitions the co-occurrence data over $\mathcal{X} \times \mathcal{Y}$ into K groups. Consider the case of supermarket shopping data, where each transaction contains a set of products and the customer who bought them. We want to find the relationships between customers and products. Let

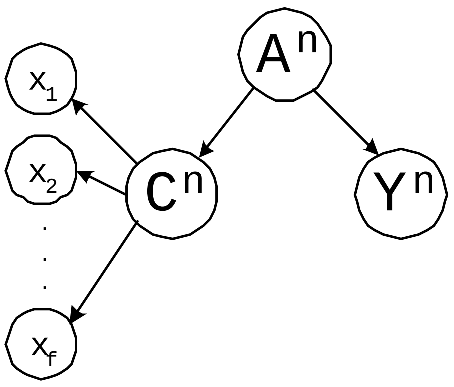
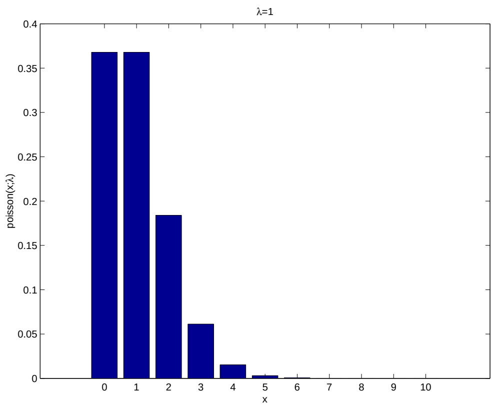


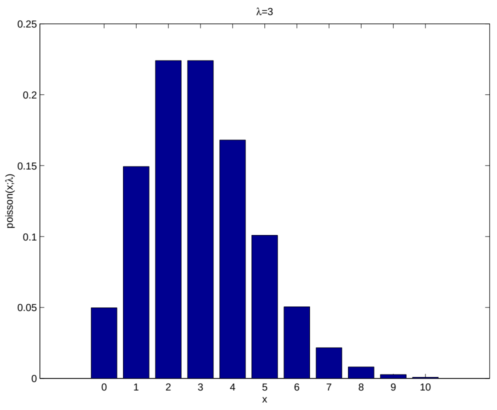
Figure 3.3: Hybrid Model

$\mathcal{X} = \{x_1, \ldots, x_I\}$ denotes the customers and $\mathcal{Y} = \{y_1, ..., y_J\}$ denotes the products. Thus the information about which customer buy which product can be represented by the pairs (x, y) that occur in the transactional data. An aspect model fits the relationship. But there is a limitation. If we have a new customer that does not appear in the data set $\mathcal{X}$, using an aspect model constructed by $\mathcal{X}$, we cannot determine the preference of this new customer. To alleviate this limitation, we assume that each customer is represented by some features and belongs to his/her related clusters resulted from the cluster model. This idea leads to the cluster/product pairs instead of customer/product pairs in the aspect model.

The complete model is a hybrid model for the situation discussed above. This model is a combination of a cluster model and an aspect model. Fig. 3.3 shows the graphical representation. The main idea is to substitute the cluster/product pair (c^n, y^n) for the original customer/product pair (x^n, y^n). Let $\mathbf{x}^n = (x_1^n, \ldots, x_f^n)$. Each data vector $\mathbf{x}^n$ has its own associated cluster $\mathbf{c}^n$ and the cluster/product (c^n, y^n) relationships are encoded in the aspect model. The input data is essentially the co-occurrence pairs $(x_1^n, \ldots, x_f^n, y^n)$ where $(x_1^n, \ldots, x_f^n)$ represents customer $\mathbf{x}^n$. By summing all the possible values of



(a) $\lambda = 1$



(b) $\lambda = 3$

Figure 3.4: Poisson distributions with $\lambda = 1$ and $\lambda = 3$

the two latent variables, C and A, we get the complete data probability

$$\begin{aligned} P(\mathcal{S}) &= \prod_{n=1}^{N} P(s^n), \quad \text{where} \\ P(s^n) &= \sum_{c}\sum_{a} P(x_1^n, \ldots, x_f^n, y^n, c^n, a^n) \\ &= \sum_{k=1}^{K}\sum_{g=1}^{G} P(x_1^n|c_g)\ldots P(x_f^n|c_g)P(c_g|a_k)P(a_k)P(y^n|a_k). \end{aligned}$$

3.2 Training HyPAM

3.2.1 Cluster Model Based on Poisson Distributions

Customers can be clustered into groups with similar shopping preferences. For example, young women like to buy cosmetics while young parents need to buy infant products. Therefore, it seems plausible to cluster customers to predict their shopping preferences. In our implementation, we choose the historical shopping lists as customers' features. Each feature is the quantity that a certain product is purchased by the customer during a given period

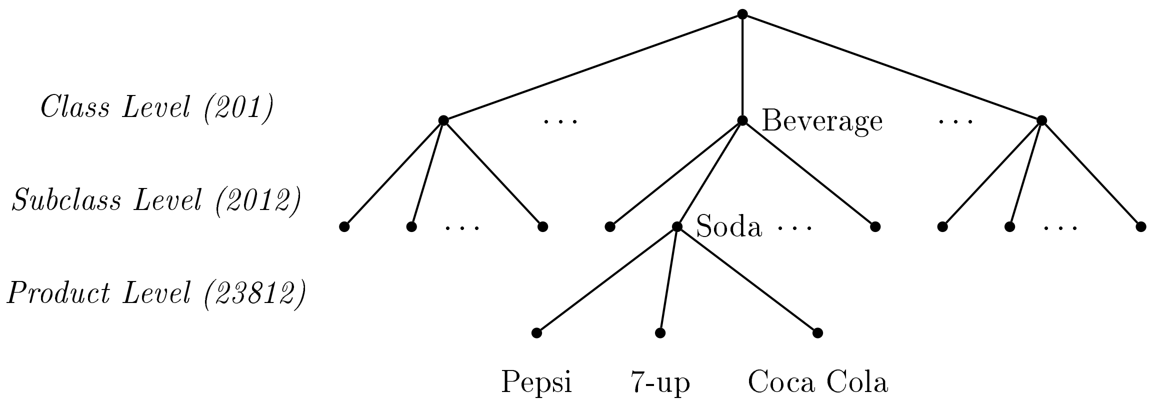


Figure 3.5: Product Taxonomy

of time. The value of such a feature is a random variable. We assume that these random variables have Poisson distributions.

Poisson Distribution The Poisson Distribution is a discrete distribution which takes on the values $X = 0, 1, 2, 3, \ldots$. It is most commonly used to model the number of random occurrences of some phenomenon in a specified unit of space or time. It can be thought as a limiting form of the binomial distribution $B(x; n, p)$ when n is very large, p is very small, and $np = E(X)$ is moderate. Applying to our shopping data modelling case, the Poisson distribution is appropriate because the probability that one may buy a certain product is very small (p is small), the number of transactions in the database is very large (n is large), and one may buy a moderate amount of a product (np is moderate).

The Poisson distribution is determined by one parameter, λ, and is given by

$$P(x; \lambda) = e^{-\lambda} \frac{\lambda^x}{x!}\ , \quad x = 0, 1, \ldots, \quad \lambda > 0$$

The expectation value and variance of a random variable X with Poisson

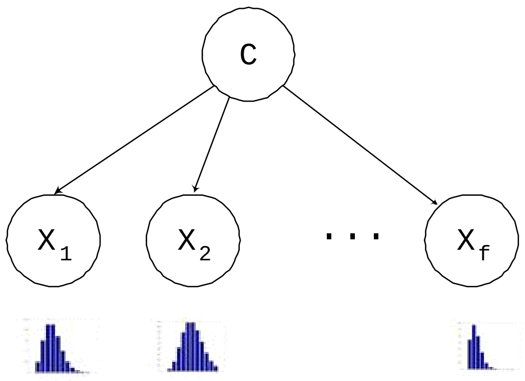


Figure 3.6: Cluster Model Based on Poisson Distributions

distribution are given by

$$E(X) = \lambda, \quad V(X) = \lambda.$$

Fig. 3.4 shows the Poisson distribution with $\lambda = 1$ and $\lambda = 3$.

Cluster Model The retail industry has adopted a standard product classification system that constitutes a hierarchical product taxonomy. For example, "Pepsi six-pack" may be classified in the "soda" subclass while "soda" is classified in the "beverage" class. Fig. 3.5 shows an example of such a hierarchy. As mentioned above, each customer is characterized by their shopping records. We may use the purchases on any level of the hierarchical product taxonomy as the features of a customer. For example, the value of the feature "soda" is the sum of the purchases of "Pepsi six-pack" and "7-up six-pack".

By assuming that each feature has a Poisson distribution, we can use a Poisson parameter vector to represent a cluster, e.g.

$$\hat{\lambda}_g = (\lambda_{g1}, \lambda_{g2}, ..., \lambda_{gf}),$$

where λ_{gf} indicates that in cluster g the average purchase of product (class) f is λ_{gf}. Fig. 3.6 is a graphical depiction.

3.2.2 Hybrid Poisson Aspect Model for Cluster/Product Relationships

In order to make personalized recommendations, we want to find the relationships between customers and products, i.e., the likelihood that a customer will buy a certain product. To model these relationships, we use a hybrid poisson aspect model (HyPAM) which models the cluster/product relationships rather than the original customer/product ones.

In HyPAM, each customer, represented by his/her historical purchases, will be assigned to a cluster with a probability as previously defined. By using the cluster model, a customer data vector $\mathbf{x}^n$ can be transformed into a probability vector $\mathbf{c}^n = (v_1^n, \ldots, v_G^n)$, where v_g^n denotes the probability of the customer belongs to cluster g. Fig. 3.7 is the graphical depiction of the complete model and its parameters.

3.2.3 Model Fitting by EM

Expectation Maximization Algorithm The Expectation Maximization (EM) algorithm [3] is a general method of finding the maximum-likelihood estimate of the parameters of an underlying distribution from a given data set when the data is incomplete with missing values.

Given some data $\mathbf{X}$ and a model parameterized by θ, the goal of EM is to find θ such that the likelihood $P(\mathbf{X}|\theta)$ is maximized. Often we maximize $\log P(\mathbf{X}|\theta)$ instead because it is analytically easier. In general, $\log P(\mathbf{X}|\theta)$ has no analytic solution. However, if we introduce some unobserved variables $\mathbf{Y}$, the maximization of $\log P(\mathbf{X}, \mathbf{Y}|\theta)$ will be much more easier. We call

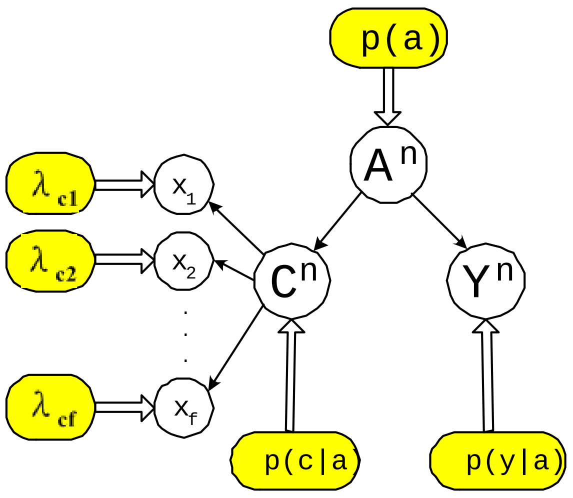


Figure 3.7: HyPAM and its parameters

$\mathbf{X}$ *observed data*, $\mathbf{Y}$ *missing data*, and $\mathbf{Z} = \mathbf{X} \bigcup \mathbf{Y}$ *complete data*. EM is an iterative optimization algorithm which defines a sequence of parameter settings through a mapping $\theta_t \to \theta_{t+1}$ such that $P(\mathbf{Z}|\theta_{t+1}) \geq P(\mathbf{Z}|\theta_t)$. Thus EM is a hill-climbing algorithm which will converge to a stationary point of $\log P(\mathbf{Z}|\theta)$.

The algorithm is defined in two steps:

1. The **Estimation** step. Define $\tilde{p}(\mathbf{Z}) = p(\mathbf{X}, \mathbf{Y}|\mathbf{X}, \theta_t)$. Calculate

$$Q(\theta', \theta_t) = E[\log P(\mathbf{Z}|\theta') \mid \tilde{p}(\mathbf{Z})] = \int \tilde{p}(\mathbf{Z}) \log P(\mathbf{Z}|\theta) d\mathbf{Z}$$

2. The **Maximization** step. Set $\theta_{t+1} = \arg\max_{\theta'} Q(\theta', \theta_t)$.

The intuition is as follows: if we had the complete data, we would simply estimate θ to maximize $\log P(\mathbf{Z}|\theta)$. But with some of the data missing, we maximize the *expectation* of $\log P(\mathbf{Z}|\theta)$ given the observed data and the current value of θ.

Fitting HyPAM Essentially, we use the co-occurrence customer/product pairs (x^n, y^n) to be the training data. As mentioned in the previous section, we use the historical purchases to represent a customer $\mathbf{x}^n$. Each instance in our training data has the form $(x_1^n, \ldots, x_f^n, y^n)$, where $(x_1^n, \ldots, x_f^n)$ is the historical purchases of customer $\mathbf{x}^n$ in the corresponding product classes. The complete data log-likelihood is

$$\begin{aligned}
\log P(\mathcal{S}) &= \sum_{n=1}^{N} \log P(\mathbf{x}^n, y^n) \\
&= \sum_{n=1}^{N} \log P(x_1^n, \ldots, x_f^n, y^n) \\
&= \sum_{n=1}^{N} \log \sum_{k=1}^{K} \sum_{g=1}^{G} P(x_1^n, \ldots, x_f^n, y^n, c_g, a_k) \\
&= \sum_{n=1}^{N} \log \sum_{k=1}^{K} \sum_{g=1}^{G} P(x_1^n|c_g) \ldots P(x_f^n|c_g) P(c_g|a_k) P(a_k) P(y^n|a_k).
\end{aligned} \tag{3.1}$$

In this case, the cluster and aspect label are the hidden variables $\mathbf{Y}$ in the framework of the EM algorithm. If we could observe the cluster and aspect labels that each customer belongs to, the complete data $\mathbf{Z}$ should be $\{(x_1^n, \ldots, x_f^n, y^n, c^n, a^n)\}_{1 \le n \le N}$, where c^n and a^n are the associated cluster and aspect labels, respectively. The EM algorithm, when applied to train a HyPAM model, is given as follows.

1. The **Estimation** step: define $\tilde{q}_{kg}^n = P(a^n = a_k, c^n = c_g|\mathbf{x}^n, y^n; \theta)$, i.e., the probability of the n'th instance being in cluster g and in aspect k given the current parameters. To simplify the notation, let $\tilde{q}_k^n = \sum_g \tilde{q}_{kg}^n = P(a_k|\mathbf{x}_n, y^n; \theta)$ and $\tilde{q}_g^n = \sum_k \tilde{q}_{kg}^n = P(c_g|\mathbf{x}_n, y^n; \theta)$. Applying Baye's rule, we have

$$\tilde{q}_{kg}^n = \frac{P(\mathbf{x}^n, y^n, a_k, c_g)}{P(\mathbf{x}^n, y^n)}$$

$$
\begin{aligned}
&= \frac{P(x_1^n, \ldots, x_f^n, y^n, a_k, c_g)}{\sum\limits_{k'=1}^{K} \sum\limits_{g'=1}^{G} P(x_{n1}, \ldots, x_{nf}, y^n, a_{k'}, c_{g'})} \\
&= \frac{P(x_1^n|c_g) \ldots P(x_f^n|c_g) P(c_g|a_k) P(a_k) P(y^n|a_k)}{\sum\limits_{k'=1}^{K} \sum\limits_{g'=1}^{G} P(x_1^n|c_{g'}) \ldots P(x_f^n|c_{g'}) P(c_{g'}|a_{k'}) P(a_{k'}) P(y^n|a_{k'})}.
\end{aligned} \tag{3.2}
$$

Now we can derive $Q(\theta', \theta_t)$ by

$$
\begin{aligned}
Q(\theta', \theta_t) &= E[\log P(\mathbf{Z}|\theta') \mid \tilde{p}(\mathbf{Z})] \\
&= \sum_{n=1}^{N} E[\log P(\mathbf{z}_n|\theta') \mid \tilde{p}(\mathbf{z}_n)] \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} \sum_{g=1}^{G} \tilde{q}_{kg}^n \log P(x_1^n|c_g) \ldots P(x_f^n|c_g) P(c_g|a_k) P(a_k) P(y^n|a_k) \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} \sum_{g=1}^{G} \tilde{q}_{kg}^n [\sum_{i=1}^{f} \log P(x_i^n|c_g) + \log P(c_g|a_k) + \log P(a_k) + \log P(y^n|a_k)].
\end{aligned}
$$

2. The **Maximization** step:

   - To obtain λ_{gi} :

     Recall that $P(x_i|c_g) = e^{-\lambda_{gi}} \frac{\lambda_{gi}^{x_i}}{x_i!}$, solving the equation:

$$
\begin{aligned}
\frac{\partial Q(\theta', \theta_t)}{\partial \lambda_{gi}} &= \sum_{n=1}^{N} \sum_{k=1}^{K} \tilde{q}_{kg}^n (-1 + \frac{x_{ni}}{\lambda_{gi}}) = 0 \\
\Longrightarrow \lambda_{gi} &= \frac{\sum\limits_{n=1}^{N} \tilde{q}_g^n x_{ni}}{\sum\limits_{n=1}^{N} \tilde{q}_g^n}.
\end{aligned} \tag{3.3}
$$

   - To obtain $P(a_k)$:

     Add a Lagrange multiplier α

$$
\frac{\partial}{\partial P(a_k)} \left[Q(\theta', \theta_t) + \alpha(\sum_{k'=1}^{K} P(a_{k'}) - 1) \right] = 0
$$

$$
\sum_{n=1}^{N} \sum_{g=1}^{G} \tilde{q}_{kg}^n \frac{1}{P(a_k)} + \alpha = 0 \quad \Longrightarrow \quad \alpha = -N
$$

$$\Longrightarrow P(a_k) = \frac{1}{N}\sum_{n=1}^{N} \tilde{q}_k^n. \tag{3.4}$$

- To obtain $P(c_g|a_k)$:

  Add a Lagrange multiplier β

$$\frac{\partial}{\partial P(c_g|a_k)}\left[Q(\theta', \theta_t) + \beta(\sum_{g'=1}^{G} P(c_{g'}|a_k) - 1)\right] = 0$$

$$\sum_{n=1}^{N} \tilde{q}_{kg}^n \frac{1}{P(c_g|a_k)} + \beta = 0 \quad \Longrightarrow \quad \beta = -\sum_{n=1}^{N} \tilde{q}_k^n$$

$$\Longrightarrow P(c_g|a_k) = \frac{\sum_{n=1}^{N} \tilde{q}_{kg}^n}{\sum_{n=1}^{N} \tilde{q}_k^n}. \tag{3.5}$$

- To obtain $P(y|a_k)$:

  Add a Lagrange multiplier γ

$$\frac{\partial}{\partial P(y|a_k)}\left[Q(\theta', \theta_t) + \gamma(\sum_{y'} P(y'|a_k) - 1)\right] = 0$$

$$\sum_{n:y^n=y}\sum_{g=1}^{G} \tilde{q}_{kg}^n \frac{1}{P(y|a_k)} + \gamma = 0 \quad \Longrightarrow \quad \gamma = -\sum_{n=1}^{N} \tilde{q}_k^n$$

$$\Longrightarrow P(y|a_k) = \frac{\sum_{n:y^n=y} \tilde{q}_k^n}{\sum_{n=1}^{N} \tilde{q}_k^n}. \tag{3.6}$$

Note that the value of some features of a customer may be missing when this customer never buy any item in the corresponding class. In these cases, just skip the missing features while computing the probability:

$$P(\mathbf{x}_n|c_g) = \prod_{i\in I_n} P(x_{ni}|c_g),$$

where I_n is the set of items that customer n bought before. While computing λ_{gi}, skip the terms $\tilde{q}_g^n x_i^n$ and $\tilde{q}_g^n$ in both denominator and numerator in Equation (3.4), if customer n never buy anything in class i.

3.3 Using the Hybrid Poisson Aspect Model

3.3.1 Recommendation

One way to solve the recommendation problem is by estimating the preferences of the items for a customer. We can make the recommendation list according to the probability $P(y|\mathbf{x})$ that customer $\mathbf{x}$ will buy product y. This quantity can also be considered as a score of the tendency or preference of customer $\mathbf{x}$ toward product y. A customer $\mathbf{x}$ can be represented by his/her historical purchases $(x_1, ..., x_p)$. Then we can easily estimate $P(y|\mathbf{x})$ from HyPAM

$$\begin{aligned}
P(y|\mathbf{x}) &= \sum_{k=1}^{K} P(y|a_k)P(a_k|\mathbf{x}) \\
&= \sum_{k=1}^{K} P(y|a_k)\frac{P(\mathbf{x}|a_k)P(a_k)}{P(\mathbf{x})} \\
&= \sum_{k=1}^{K} P(y|a_k)\frac{P(a_k)}{P(\mathbf{x})}\sum_{g=1}^{G} P(\mathbf{x}|c_g)P(c_g|a_k) \\
&\propto \sum_{k=1}^{K}\sum_{g=1}^{G} P(\mathbf{x}|c_g)P(c_g|a_k)P(a_k)P(y|a_k), \qquad (3.7)
\end{aligned}$$

where $P(\mathbf{x}|c) = P(x_1|c)\cdots P(x_p|c)$.

3.3.2 Inference

In addition to estimating $P(y|\mathbf{x})$, we may ask a few queries to HyPAM to support marketing decision making. Here are some examples:

- What is the distribution of the clusters?

$$P(c_g) = \sum_{k=1}^{K} P(c_g|a_k)P(a_k)$$

- What is the most popular product?

$$\arg\max_y P(y) = \arg\max_y \sum_{k=1}^{K} P(y|a_k)P(a_k)$$

- What are the most popular products in each cluster?

$$\begin{aligned}\arg\max_y P(y|c_g) &= \arg\max_y \frac{P(y,c_g)}{P(c_g)} \\ &= \arg\max_y \frac{\sum_{k=1}^{K} P(y|a_k)P(c_g|a_k)P(a_k)}{\sum_{k=1}^{K} P(c_g|a_k)P(a_k)}\end{aligned}$$

Chapter 4

Experimental Evaluation

In this chapter, we present our experimental evaluations on a set of real-world shopping data. First, we give some description and statistics of the data set in 4.1. Then in 4.2, we show two measurements to evaluate different recommendation systems. Finally, we compare HyPAM with two other approaches, GroupLens and IBM, described in Chapter 2. As we will see, HyPAM outperforms these approaches in this experimental evaluation.

4.1 Data Set

The data used in our experiments is the courtesy of a local retail supermarket for our research. This data set contains customer shopping records collected in a time span of four months, from November, 2000 to February, 2001. Each record in the data set consists of four attributes: the shopping date, customer ID, product number, and the amount of purchase. Shopping records with the same customer ID and the same shopping date are considered as a *transac-*

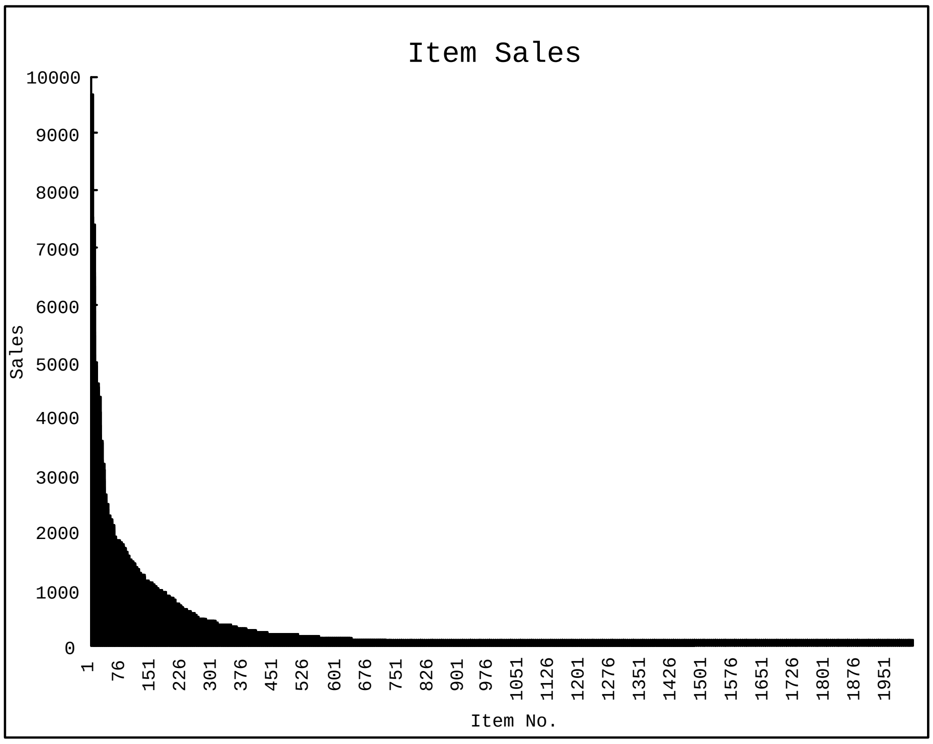


Figure 4.1: Data skewness

tion. There are 119,578 transactions and 32,266 distinguishable customers appear in this data set. The average purchase of items in each transaction is 17.76 with a 19.08 standard deviation. The sale figure is skew and concentrated on a very small portion of items. This is an example of "*the 80-20 rule*" known in business management. The 80-20 rule originally came from an Italian economist called Vilfredo Pareto (1848-1923) who, in 1906, observed that 20% people owned 80% of accumulated wealth in Italy. The 80-20 rule has been applied in many management problems and in this case, predicts that 80% of sales come from 20% of items. Fig. 4.1 shows this phenomenon. As a result, association rules may not be predictive because items appear in high support association rules may only come from the small portion of popular items while most items will be neglected by association rule mining

(a) Subclass Level

Item1	Item2	Support	Confidence
500117	500102	11	0.92
470609	470202	41	0.89
760576	760577	23	0.85
120111	130305	10	0.83

(b) Class Level

Item1	Item2	Support	Confidence
7311	7307	11	0.65
3212	3213	10	0.59
3009	5002	10	0.56
4706	4702	129	0.55

Table 4.1: Two-item sets sorted by confidence

algorithms. Table 4.1 (a) and (b) show the the top four highest confidence association rules of two-item sets with support ≥ 10 at the subclass and class levels, respectively. Table 4.2 (a) and (b) show the top four highest support association rules of two item sets with confidence ≥ 0.4 at the subclass and class levels, respectively.

In addition to the data set, we also exploit a three-level hierarchical product taxonomy. Products are divided into 201 product classes. Each product class is subdivided into less than 100 subclasses, generating a total 2,012 product subclasses. Absolute customer spending is available from the data at any level in this hierarchy. Fig. 3.5 illustrates a portion of the hierarchical product taxonomy.

We randomly chose 15,000 customers' shopping records to be the training data and 1,000 customers from the remainder for testing. The purchases that we deal with are on the subclass level. To learn the model parameters, we converted the records in the training data into the form $\{\mathbf{x}^n, y^n\}_{1\leq n\leq N}$, where N is the total number of these customer/product co-occurrence data and y^n is the subclass ID of this product. The first element $\mathbf{x}^n = (x_1^n, \ldots, x_p^n)$ indicates that customer n bought products in subclass 1 to subclass p during the four

(a) Subclass Level

Item1	Item2	Support	Confidence
100312	100205	3026	0.45
530103	530101	1360	0.43
100217	100205	912	0.44
100218	100205	889	0.47

(b) Class Level

Item1	Item2	Support	Confidence
1003	1002	12398	0.46
1002	1003	12398	0.44
1104	1101	9594	0.42
5301	5002	9052	0.41

Table 4.2: Two-item sets sorted by support

months period, and the value of x_i^n represents the amount of purchase. In this case, $N = 119,578$ and $P = 2,012$.

4.2 Evaluation Methodology

In order to evaluate our approach, we adopt two metrics, *rank score* and *lift index.* These two metrics return a score that measures the quantity of a given ranked list generated by a recommender system with regard to a customer and a list of items that the customer actually purchased. We use the *Given n* and *All but one* protocol to divide a shopping list into training and test items while evaluating different approaches with the two metrics. while calculating the values of the two metrics.

$\mathbf{R_a}$	
300206	Hit
300203	Hit
300210	
300207	
300208	Hit
300209	
⋮	⋮

$\mathbf{R_{max}}$	
300206	Hit
300203	Hit
300208	Hit
⋮	⋮

$\alpha = 10$

$$R_a = \frac{1}{2^{\frac{1-1}{10-1}}} + \frac{1}{2^{\frac{2-1}{10-1}}} + \frac{1}{2^{\frac{5-1}{10-1}}} + \cdots$$

$$R_{max} = \frac{1}{2^{\frac{1-1}{10-1}}} + \frac{1}{2^{\frac{2-1}{10-1}}} + \frac{1}{2^{\frac{3-1}{10-1}}} + \cdots$$

Figure 4.2: Example of Calculating Rank Score

4.2.1 Rank Score

Breese et al. [2], originally for a Web page recommendation task, defined the expect utility of a ranked list of items as

$$R = \frac{R_a}{R_a^{max}}, \tag{4.1}$$

$$R_a = \sum_j \frac{\delta(a,j)}{2^{(j-1)/(\alpha-1)}}, \tag{4.2}$$

where j is the rank of an item in the full list of suggestions proposed by a recommender, $\delta(a,j)$ is 1 if user a accessed item j in the test set and 0 otherwise, and α is the *viewing half-life*, which is the place of an item in the list such that it has a 50% chance of being viewed. We use $\alpha = 10$ in our experiments. The expected utility R is normalized by R_a^{max}, which is the maximum possible utility obtained when all items that user a has accessed appear at the top of the ranked list. Fig. 4.2 is an example of how to calculate rank score. Note that if the recommender considers that the customer has the same preference of more than one items, then these items will have the same rank in the suggestion list.

4.2.2 Lift Index

Another evaluation metric is *lift index* [6]. After the recommender generates the ranked list, we divide the list into 10 equal deciles, and see how the items accessed by the user in the test set distribute in the 10 deciles. Clearly, the more items appear in the top deciles, the better this ranked list is. The lift index is defined as the weighted sum of the number of accessed items appear in the 10 deciles. Assuming the number of the accessed items in each decile (ordered) are $S_1, S_2, \ldots, S_{10}$, the *lift index* is defined as

$$S_{lift} = \frac{1 \times S_1 + 0.9 \times S_2 + \ldots + 0.1 \times S_{10}}{\sum_i S_i}.$$

Here is an example distribution of the accessed items in the deciles:

S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
10	7	3	0	2	0	0	0	0	0

Then the lift index is

$$S_{lift} = \frac{10 \times 1 + 7 \times 0.9 + 3 \times 0.8 + 2 \times 0.6}{10 + 7 + 3 + 2} = 0.90.$$

In the best situation, when $S_1 = \sum_i S_i$, $S_{lift} = 1$. In the random case, the items distributed uniformly in all deciles, $S_{lift} = 0.55$.

$$\begin{aligned} S_{lift(random)} &= \frac{1 \times S_r + 0.9 \times S_r + \ldots + 0.1 \times S_r}{\sum_i S_r} \\ &= \frac{S_r \times (1 + 0.9 + \ldots + 0.1)}{10 \times S_r} \\ &= 0.55. \end{aligned}$$

As mentioned above, two items will have the same ranks if the recommender considers that the customer has the same preference of these two items.

4.2.3 Protocol

In our experiments, we adopted different *protocols* to divide the shopping list into training and test sets to simulate situations with differing numbers of purchases available to the recommenders [2]. In the first protocol, we withheld a single randomly selected purchase for each user, and the remainder purchases are treated as the observed purchases to be checked in the ranked list. This protocol is called *All but one.* Other protocols is to randomly select 2, 5, or 10 purchases from each test customer as the observed purchases, and check the remainder in the ranked list. They are referred to as *Given n* protocols.

The *All but one* protocol measures the algorithms' performance when given as much data as possible for each test customer. The various *Given n* protocols examine the performance of the algorithms when there is relatively little known purchases about an active customer. Note that in the experiment applying *Given n* protocol, only those customers with more than n purchases will be selected in the test set of 1000 customers.

4.3 Comparison

In this section, we present the experimental results of the three approaches, GroupLens, IBM, and HyPAM. In order to conduct the experiments, we transformed the original data into the form *(Customer ID, Subclass No., Amount).* This data preprocessing step was accomplished using some simple SQL commands. For each approach, we randomly selected data of 15,000 customers to train the model, and used the data of another 1,000 customers

Table 4.3: Experimental Result—Rank Score

Algorithm	Given 2	Given 5	Given 10	All but 1
GroupLens	0.0276	0.0285	0.0324	0.0298
IBM	0.0976	0.0944	0.0715	0.132
C10_A10	0.279	0.275	0.281	0.280
C10_A20	0.275	0.274	0.276	0.277
C20_A10	0.279	0.271	0.286	0.278
C20_A20	0.275	0.274	0.276	0.278

Training set size: 15000   Test set size: 1000

to test the accuracy of the recommendation generated by the trained model. The accuracy is then measured by the combinations of the two metrics under the *Given n* and *All but one* protocols, as described in Section 4.2. Table 4.3 and Table 4.4 show the experimental results of rank scores and lift index, respectively. In each talk, C10_A10 represents the result of HyPAM using 10 clusters and 10 aspects, and so on. The results show that HyPAM outperforms the other two approaches significantly in all combinations of experimental setting.

Table 4.4: Experimental Result—Lift Index

Algorithm	Given 2	Given 5	Given 10	All but 1
GroupLens	0.877	0.870	0.866	0.886
IBM	0.809	0.824	0.826	0.804
C10_A10	0.954	0.953	0.953	0.953
C10_A20	0.954	0.953	0.953	0.953
C20_A10	0.953	0.953	0.953	0.953
C20_A20	0.954	0.953	0.953	0.953

Training set size: 15000   Test set size: 1000

We also compared the computation time of these approaches. For GroupLens, there is no training time required. For IBM method, the training time is the sum of the time spent for constructing customer and product models [5]. To construct the product model, we have to generate the association rules of two-item sets. It took about 30 minutes to compute the algorithm of the association rule mining. The association mining algorithm we used is taken from [1]. By using some SQL commands, we construct the customer model in another 3 minutes. So the training time of IBM method is about 33 minutes. For HyPAM, the training time is spent on learning the model parameters. It ranges from 5 hours to 11 hours depending on the number of clusters and aspects. Table 4.5 shows the training and testing time of these three algorithms. The testing time is the time of generating one ranked list for a customer.

From Table 4.3, we can see that the rank score of HyPAM is about three times as good as IBM and ten times as good as GroupLens. The lift index of HyPAM is also better than the others. We also observed that both rank score and lift index are not sensitive to the number of clusters and aspects of HyPAM. The results of the combinations of 10 or 20 clusters and aspects are almost the same. Using 10 clusters and 10 aspects is good enough (with less training time). Although the training time of HyPAM is quite long, the testing time of HyPAM is much shorter than the others. For applications where on-line generation of recommendation for a new customer is required, and the model can be constructed off-line, such as those in Web-based B2C merchants, HyPAM clearly holds the advantages.

Table 4.6 to 4.8 give the experimental results of three individuals who purchased items from a total of 9 product subclasses. The numbers listed on

	GroupLens	IBM	HyPAM
training time	0	33 min	5 ∼11 hr
testing time	2.5 sec	10 sec	0.4 sec

Table 4.5: Training and Testing Time

the table are the positions of purchased products on the ranked list. These experimental results are all generated using *Given 5* protocol and thus there are four product subclasses in each table.

Table 4.6: Example(1): Recommendation Result of an individual

	GroupLens	IBM	C10_A10
Hit Position in the list	42	43	1
	76	88	17
	230	151	28
	349	229	251
Rank Score	0.018	0.015	0.410
Lift Index	0.95	0.975	0.975

Customer: 00001069   Total Buy: 9   Protocol: Given 5

Table 4.7: Example(2): Recommendation Result of an individual

	GroupLens	IBM	C10_A10
Hit Position in the list	129	27	14
	176	33	45
	235	55	108
	694	1537	270
Rank Score	4.3e-5	0.08	0.13
Lift Index	0.93	0.975	0.975

Customer: 00009218 Total Buy: 9 Protocol: Given 5

Table 4.8: Example(3): Recommendation Result of an individual

	GroupLens	IBM	C10_A10
Hit Position in the list	13	42	6
	297	85	34
	341	1352	155
	701	1479	608
Rank Score	0.129	0.017	0.240
Lift Index	0.900	0.925	0.975

Customer: 00019484 Total Buy: 9 Protocol: Given 5

Chapter 5

Conclusion

In this thesis, we propose a probability model called *Hybrid Poisson Aspect Model* (HyPAM), which is a combination of cluster and aspect models. This model is constructed for personalized shopping recommendations. By combining a cluster model with an aspect model, we can alleviate the limitation of most collaborative filtering approaches that the active user must be in the training set. We can give recommendations to a customer easily by providing the system the customer's historical purchases. As we saw in Chapter 4, HyPAM outperforms the two well-known recommender systems, GroupLens and IBM method. The prediction time is quite short that we can apply HyPAM to the real time use, such as an e-commerce Web merchant where a large amount of requests may arrive in a very short period of time.

However, HyPAM takes much time to complete training. To alleviate this problem, two approaches can be taken. First, we can parallelize the EM algorithm and execute the training algorithm on a PC cluster. Second, our colleagues are developing a novel convergence test based on the maximum free storage principle, which combines maximum likelihood and maximum entropy principles to speed up the convergence and avoid overfitting for the EM algorithm.

Bibliography

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94)*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[2] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI '98)*, pages 43–52, 1998.

[3] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal statistical Society*, B39:1–37, 1977.

[4] Thomas Hofmann and Jan Puzicha. Latent class models for collaborative filtering. In *Proceedings of the International Joint Conference in Artificial Intelligence*, pages 688–693, 1999.

[5] Richard D. Lawrence, George S. Almasi, Vladimir Kotlyar, Marisa S. Viveros, and Sastry Duri. Personalization of supermarket product recommendations. *Data Mining and Knowledge Discovery*, 5:11–32, 2001.

[6] C.X. Ling and C.Li. Data mining for direct marketing: Problems and solutions. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 73–79. AAAI Press., 1995.

[7] P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina, 1994.

[8] Upendra Shardanand and Patti Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, pages 210–217, 1995.