Competitive Neural Networks for Customer Choice Models

Walter A. Kosters and Michiel C. van Wezel

Leiden Institute of Advanced Computer Science
Universiteit Leiden
P.O. Box 9512, 2300 RA Leiden, The Netherlands
Email: {kosters,michiel}@liacs.nl

Abstract. In this paper we propose and examine two different models for customer choices in for instance a wholesale department, given the actual sales. Both customers and products are modeled by points in a k-dimensional real vector space. Two possible strategies are discussed: in one model the customer buys the nearest option from categories of products, in the other he/she buys all products within a certain radius of his/her position. Now we deal with the following problem: given only the sales list, how can we retrieve the relative positions corresponding to customers and products? In particular we are interested in the dimension k of the space: we are looking for low dimensional solutions with a good "fit" to the real sales list. Theoretical complexity of these problems is addressed: they are very hard to solve exactly; special cases are shown to be NP-complete. We use competitive neural network techniques for both artificial and real life data, and report the results.

1 Introduction

Often shop owners know very little about their customers. We examine the situation where only sales slips are present, i.e., for every customer visiting the shop a list of his or her purchases is available. It is obvious that these data contain very valuable information, which could lead to interesting insights. The question we ask ourselves here is how many "underlying" dimensions do exist that influence the client in his/her decision when purchasing certain goods. Marketing literature suggests that such underlying dimensions indeed exist, see [11]. In the example of a wholesale department with an extensive catalogue of products, two such dimensions might be price and brand-quality.

We propose different models to analyse (and hopefully understand) the behaviour of the customers. In both models customers and products are modeled as points in a k-dimensional real vector space. These k dimensions are supposed to represent the underlying dimensions mentioned in the previous paragraph. Without using any interpretation of these dimensions, we try to find the number of relevant dimensions and the coordinates of customers and products. Contrary to many approaches (cf. [4]) our methods use only very partial information; instead of direct attributes like age or price, we only use the sales list. Notice that the identity of the customers is entirely hidden: only

the sales slips are used; if a customer appears for the second time, he/she is even considered to be a new customer.

In the first model, the products are grouped into a small number of categories. Every customer has to buy a product from each category, choosing from the options (products) available in it. The customer chooses—from the different options within each category—the nearest one. In the second model every customer buys exactly those products that are within a certain distance, the so-called radius of the customer. Some notion of distance is required; we examined the Euclidean distance and some Manhattan-like distances.

Now the problem can be easily described. Given a list of sales slips, determine a dimension k such that the model generates those same purchases—up to a certain error. Here k should be as small as possible. Furthermore, we also want to find the coordinates of both customers and products. The models and the exact problem will be described in Section 2 and Section 3.

In Section 4 we will show (i.e., give a proof outline) that the problem is already NP-complete (see [5]) in special cases. This means that an exact solution is beyond reach, possibly for many years to come. This theoretical complexity justifies the use of approximating algorithms, such as neural networks (see [9,2]). In Section 5 we describe the neural networks that we have used in an attempt to find a satisfactory solution to this problem. They are related to simple competitive neural networks.

In Section 6 we provide some results of experiments on artificial data, using those neural networks. We pay special attention to the dimension of the space, and explain a method to find the "true" dimension k. Customer and product points in k-space were generated randomly, and sales were generated according to one of the models. Throwing away the original points, the problem was to retrieve their coordinates. Experiments show that this is feasible, especially in the case where every customer buys lots of products.

Exploratory experiments on real data are reported in Section 7. Real data offer more difficulties than artificial data. Motivated by the results on artificial data we restricted the experiments to those customers who bought at least 40 products. At first it seemed hard to improve upon the so-called naive error: the system that buys nothing at all has a relatively small error. Fortunately we had several runs that were promising. The experimental results can be analysed by, e.g., cluster analysis in order to obtain a better understanding of the coordinates—and the dimensions.

Finally, in Section 8, we discuss the techniques used, and we mention some practical applications. In order to use these techniques, it is only necessary to have the sales lists available. In case of the option model, a field expert has to divide the products into categories. The results have to be interpreted by experts, but can also be used directly.

Part of the work presented in this paper was published in [15] and [10]. More details on the NP-completeness issue can also be found in the second author's forthcoming PhD thesis.

2 The Models and the Data

Suppose we are given n different customers $x_1, x_2, \ldots, x_n$ and m different products $p_1, p_2, \ldots, p_m$. We assume that customers buy each product at most once. Customers buying exactly the same products are identified. We have to analyse the so-called sales list: for every customer we know exactly the products he or she bought. We now propose two models to understand the behaviour of the customers using embeddings in a real vector space.

In Fig. 1 we see a small example with $n = 2$ and $p = 6$. The embedding is realised in a 2-dimensional space.

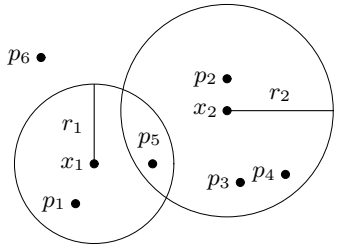

Fig. 1. Example of an embedding with 2 customers and 6 products. The spheres are only part of the product model.

2.1 The Option Model

The *option model* was introduced in [15]. In this model, the products are divided into c disjoint categories $C_1, C_2, \ldots, C_c$. In each category we have a number of products, say $c_\ell \geq 2$ in category C_ℓ ($\ell = 1, 2, \ldots, c$); these products are called the *options* or the *alternatives*, hence the name option model. Note that $\sum_{\ell=1}^{c} c_\ell = m$.

Every customer x_i has to choose exactly one product from every category. If both customers and products are represented by points in a k-dimensional vector space, the customer is supposed to choose (for every category) the option that is nearest to him/her. We assume that there always is exactly one option that is the nearest; if not, we slightly disturb the customers' coordinates. Here we need a certain metric on the space.

For the example in Fig. 1 we can imagine a situation where $c = 2$, and the first category consists of products p_1, p_2, p_3 and p_4, whereas the second category consists of products p_5 and p_6. Here customer x_1 will buy p_1 and p_5, and customer x_2 will buy p_2 and p_5.

It is also possible—but not obvious—to deal with the concept of non-buying in this model: an extra option could be added, corresponding to the

situation where the customer does not want to choose from the original options. Finally, the division into categories is in principle done by the wholesale department.

2.2 The Product Model

Secondly, we describe the *product model*. It was introduced in [10]. Here, each customer x_i also has a *radius* r_i ($i = 1, 2, \ldots, n$), proportional to the number of products he or she bought. Again, both customers and products are represented in our model by k-dimensional vectors. There is no subdivision of products into categories anymore. In this model a client buys all products within his/her own radius.

In Fig. 1 the radiuses r_1 and r_2 of the spheres are chosen in such a way that customer x_1 will buy p_1 and p_5 (as above), but customer x_2 will buy p_2, p_3, p_4 and p_5.

2.3 The Data

In our analyses the sales lists consisted of lines with numbers: each line represented one customer and each number represented one product.

Note that a sales list that is generated by customers behaving according to the option model is of a different type than one generated by customers behaving according to the product model, because in the latter case there is no subdivision into categories. This means that the type of sales lists we can analyse is different for both models. As an example, consider the following hypothetical supermarket sales list, where names have been added:

```
john      bananas    milk
mary      donuts     milk
harry     donuts     orange_juice
george    beer       orange_juice
```

If we assume that beverages form one category, this sales list cannot be generated by customers behaving according to the option model, because George buys two beverages, while in the option model the alternatives within one category exclude one another. Using the product model, this sales list could occur. Also note that in the option model every customer buys the same number of products.

3 Description of the Problem

Given an embedding of customers and products, the corresponding so-called virtual sales list can easily be constructed. To do so, we just have to determine the products that each customer will buy according to the model and the embedding. We can either use the option model or the product model as a

model for the customer behaviour when constructing this virtual sales list. We denote the real sales list by RS, and the virtual sales list by VS. If customer x_i buys product p_j in sales list RS, we say that $(x_i, p_j) \in RS$; similarly, we can define $(x_i, p_j) \in VS$.

Now the problem is the following. Given RS, try to find an embedding such that VS resembles RS as much as possible, and such that the dimension k is as low as possible. In fact, it can be shown that if $k = m$ a zero error solution can be given. To do so for the option model, take the vector p_j $(j = 1, 2, \ldots, m)$, corresponding to product j, to be $(0, \ldots, 0, 1, 0, \ldots, 0)$ (a one in coordinate j), and just add the vectors for the products bought in reality to produce the vector corresponding to customer i $(i = 1, 2, \ldots, n)$. However, in order to hope for some real life interpretation we are interested in situations where k is in the order of magnitude 5 to 10 or even lower.

The two-sided error that is used for the evaluation of VS with respect to RS can be defined—for the moment—as the number of products/options bought by the model, but not by the customers (denoted by E_1), added to the number of products/options bought by the customers, but not by the model (denoted by E_2). More precisely, the total error E is

$$E = E_1 + E_2 = \sum_{i=1}^{n} \sum_{j=1}^{m} \{E_1(i,j) + E_2(i,j)\} ,$$

where

$$E_1(i,j) = \begin{cases} 1 \text{ if } (x_i, p_j) \in VS \text{ and } (x_i, p_j) \notin RS \\ 0 \text{ otherwise} \end{cases}$$

and

$$E_2(i,j) = \begin{cases} 1 \text{ if } (x_i, p_j) \in RS \text{ and } (x_i, p_j) \notin VS \\ 0 \text{ otherwise.} \end{cases}$$

So the naive error, where the system buys nothing at all, equals the total number of sales; in this case $E_1 = 0$.

4 Complexity of the Problem

In this section we shall examine a special case of the option model, and we shall prove that the corresponding decision problem is NP-complete in the sense of [5]. These problems are very difficult indeed; up to this moment nobody has been able to find efficient solutions to any of them. Often approximating or probabilistic algorithms are used, for instance neural networks. Since exact solutions are not feasible, one would be happy to accept near optimal ones. A famous example is the Traveling Salesman Problem, where a person is asked to visit a given number of cities using a route as short as possible. NP-complete problems are usually formulated as decision problems with a yes/no answer. The most common technique used to show NP-completeness is called reduction, where a known NP-complete problem is

reduced—in a very precise way—to the problem at hand. For instance, in [5] a reduction is given from the Traveling Salesman Problem to the problem of finding a Hamiltonian circuit in a graph: a closed route connecting all nodes, visiting them once.

This section is meant for those who are interested in the theoretical background, and can be skipped on first reading. Instead of giving all details of proofs, we only provide some ideas underlying them. More details can be found in the second author's forthcoming PhD thesis. There the situation for the product model is also dealt with.

We first describe the special case of the option model we would like to address. We have n customers, who must choose from only $c = 2$ categories, with a total of m products. We identified customers who had bought exactly the same products. The customer choices are easily represented using a graph consisting of two horizontal rows of nodes, where the nodes in the first row correspond to the options from the first category, and the nodes in the second row to the options from the second category. The edges now correspond to the choices made: every customer corresponds to a unique edge. So the graph has m vertices and n edges; we call it a *sales graph*. We may assume that this graph is connected. An example, with $m = 6 + 5 = 11$ and $n = 10$, is shown in Fig. 2.

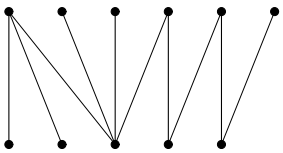

Fig. 2. Example of a sales graph with 10 customers and $6 + 5 = 11$ products.

A sales graph is *bipartite*: the nodes can be split into two nonempty disjoint sets, such that every edge is incident with a node from both sets. In fact, the partition corresponds to the categories. The converse also holds: every bipartite graph can be viewed as a sales graph.

Now we return to our original problem, and we consider the special case of dimension $k = 1$: e.g., "we only look at the price of the products". Is it possible to find an exact solution in this case? In other words, is it possible to attach real numbers to both customers and options (where customers choose the nearest option), in such a way that the graph represents the corresponding real sales? In dimension 1 we should divide the real axis for both categories into disjoint intervals, the product points lying in the centres of these intervals; in higher dimensions we get Voronoi cells here.

One can infer that the two subgraphs from Fig. 3 give rise to problems—and these are the only ones. If at least one of them occurs as a subgraph in the given graph, it is impossible to find the proper coordinates.

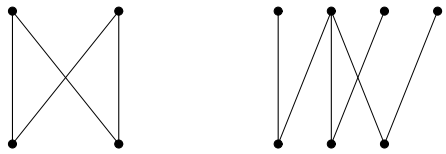

Fig. 3. Two forbidden subgraphs.

For instance, for the left hand side graph we have to cut the real axis into two parts—for both categories—in such a way that all four combinations have a nonempty intersection. This is impossible. In every way we choose the coordinates for customers and options, at least one customer will be treated incorrectly. In fact, if for the first category, with options A and B, we choose real numbers a and b with $a < b$, and for the second category, with options C and D, we choose real numbers c and d with $c < d$, then a customer corresponding to x chooses A if $x < (a + b)/2$ and B if $x > (a + b)/2$, and similarly for C and D. Now—given real a, b, c and d—it is not possible to find x_1 that buys both A and D, *and* x_2 that buys both B and C.

In general we see that cycles are forbidden, in other words: the sales graph should be acyclic, i.e., not contain any paths from a node to itself. It seems that in the example it could be possible to deal with this problem by allowing nondeterminism, for instance with x's precisely in the middle of a and b. However, first of all the model would become more complex, and secondly this would also fail in the case of cycles with more than four nodes.

Since crossing edges correspond to the fact that customers are treated correctly or not, they should be avoided. We can conclude that a sales graph can be realised in the way we want, if and only if it contains no cycles and no subgraphs isomorphic to the right hand side graph in Fig. 3—let us call it $\mathcal{H}$. The minimal number of edges that should be omitted in order to obtain a graph that meets the conditions, is exactly the minimal number of customers that is treated incorrectly.

So we are motivated to examine the following decision problem, called SG (Sales Graph). Given a bipartite graph $\mathcal{G}$; does $\mathcal{G}$ possess a spanning tree (a connected subgraph without cycles, containing every node) that does not have a subgraph isomorphic to $\mathcal{H}$? Stated otherwise, does $\mathcal{G}$ have a spanning tree that looks like a long "spine", to which several degree one nodes are attached?

We now arrive at the main result of this section:

Theorem SG *is* NP-*complete.*

The proof proceeds by reducing problem GT39 from [5] to SG. This problem is: given a bipartite graph $\mathcal{G}$; does $\mathcal{G}$ possess a Hamiltonian path, i.e., a path connecting all nodes without repeated visits to nodes?

Now that we know that SG is NP-complete, we can infer that given customers and their choices, it is not feasible to determine the minimal error that can be reached. If this were possible, the existence of a spanning tree without subgraphs isomorphic to $\mathcal{H}$ would be immanent. It is therefore appropriate to turn to approximating algorithms, such as neural networks.

5 The Neural Network Used

The neural networks we used in our experiments were inspired by simple competitive neural networks (see, e.g., [8,9]). In such neural networks, the weight vectors associated with the neurons are merely points in a k-dimensional real vector space if the input data for the network are k-dimensional real vectors. Each time a data vector is presented to a competitive neural network, a "winning unit" w is determined by calculating which unit has its weight vector closest to the presented pattern (in a Euclidean sense). Next, this units' weight vector is moved towards the presented pattern. This increases the chance of unit w being the winner if the same input pattern is presented again later on. This type of neural network is also known as a "winner-take-all-network" (see [9,6,7]). These networks perform a similar task as the classical k-means clustering methods (see [13]).

In our case, where we analyse sales lists, all customers and all products (or options within a category) are represented by a separate competitive unit. The weight-vectors of the units correspond to the points in a k-dimensional real vector space describing the customers and products (or options).

Of course, the learning process has to be modified in order to be able to work with sales lists. The aim of the learning process is to bring weight vectors of customers and weight vectors of the products bought sufficiently close to each other, whereas weight vectors of the products the customer has not bought should be at a sufficient distance.

The meaning of the word sufficient is crucial here, and it is determined by the model used. In the case where a customer buys everything within a certain radius of his or her own weight vector, sufficiently close means within this radius, and sufficiently distant means outside this radius. In the case where a customer buys the closest one of a number of options within a category, sufficiently close means closer than all the other options in the category, and sufficiently distant means at a greater distance than the option the customer *did* buy from the category.

5.1 Training Procedure for the Option Model

The training of our network in the case of the option model roughly proceeds as follows. First, the network weights are randomly initialized in the $[0;1]^k$ hypercube. Next, for a predetermined number of iterations the clients and the purchases they made are presented to the network once in random order. If client x_i chooses product p_j, the weight vectors of the units representing x_i and p_j were pulled together a little bit. This way, we hoped to increase the probability that the units representing the products that the customer has chosen, are the closest units within each product-category in the next iteration.

The parallel with a simple competitive neural network should be clear. In a simple competitive neural network the weight vectors of the units are pulled towards the input vectors. This way a Voronoi tessellation of the input space is created. In our case, we know in advance that the alternatives within one set divide the input space in a Voronoi tessellation. We also know which client should lie in which Voronoi cell for every product category in the choice process. By means of the "competitive learning" algorithm, we move the positions of the clients and the alternatives around, and we hope to reach a state where most of the clients lie in the correct Voronoi-cell for most of the product-categories in the choice-process.

Unfortunately, there is a problem associated with this learning-scheme. If we start with random initial weights, the average direction of the weight updates will be inward. This will cause the neural network to "implode". We can prevent this by re-normalising the weight vectors after each iteration.

There is another potential pitfall for our system. Typically, we have data on several thousands of customers, but there are only a few product categories, with a few alternatives within them. This causes the alternative coordinates to be updated much more frequently than the client coordinates. This problem can be solved by updating the weight vectors of the units representing the alternatives by a much smaller amount than the weight vectors of the units representing the customers.

5.2 Training Procedure for the Product Model

The training of our network in the case of the product model is very similar and roughly proceeds as follows. First, the network weights are randomly initialized in the $[0;1]^k$ hypercube. Next, the network is trained for a number of iterations (called cycles). In each iteration, a number (say 5000) of random customer/product pairs are examined. If the customer and the product are not sufficiently close, the weight vectors representing them are pulled towards each other in the k-dimensional hypercube. If they are not sufficiently far apart, they are pushed away from each other. The amount by which the weight vectors are altered is determined by a learning rate parameter. At the end of an iteration the coordinates are renormalised in order to make

optimal use of the full unit cube. Iterating is stopped if the error (which is described below) has not decreased anymore for a sufficiently large number of iterations.

At the end of each run we in particular have three significant figures:

"Both bought": number of purchases that are correctly modeled.
"Only model bought": number of purchases only performed in the model, but not in reality; this is E_1.
"Only reality bought': number of purchases only performed in reality, but not in the model; this is E_2.

As we remarked before, the error E is defined as the sum of "only reality bought" and "only model bought": $E = E_1 + E_2$. Note that the sum of "both bought" and "only reality bought" always equals the total number of sales.

6 Results on Artificial Data

In this section we present the results we obtained in several experiments with artificial data. The artificial data were generated by randomly assigning values in the $[0; 1]^k$ hypercube to the weight vectors representing a set of customers and a set of products. Subsequently, the product model or the option model was used to generate a sales list. Thus, these sales lists were generated by customers known to behave exactly as the model specifies. After generating these sales lists the customer and product embedding coordinates were thrown away, and it was attempted to recover them (or at least the relative coordinates) using the neural networks described in Section 5.

The experiments that were performed for the option model were aimed towards finding the best dimension for the embedding space (as reflected in the title of [15]). The results for the product model are more general.

6.1 Results on Artificial Data Using the Option Model

As stated above, the experiments with the option model neural networks were geared towards finding the "best" embedding dimension for a given sales list. We hope that the dimensions we find can be interpreted by a domain expert, revealing hidden motives underlying customer behaviour.

In Section 5 we explained how the neural networks could be used for finding embedding coordinates. During this discussion it was assumed that the correct embedding dimension k was given. However, in real situations k is unknown. How should we determine the best value for k?

This problem can be solved by making a "fit vs. number of dimensions"-plot. As fit measure for this plot we use the error function E from Section 3. Now we can construct a graph by running the neural network with dimension $D = 1, \ldots, N + 5$, where N is an educated guess about the true number of underlying dimensions. This "fit vs. number of dimensions"-plot will show an

elbow at the right number of dimensions. An example of a "fit vs. number of dimensions"-plot clearly showing an elbow is given in Fig. 4.

A very similar procedure for obtaining the number of underlying dimensions is often used in Multidimensional Scaling (MDS; see, e.g., [3]) and Principal Component Analysis (PCA; see, e.g., [12]). In the context of the latter technique, the "fit vs. number of dimensions"-plot is often called "scree plot".

In total we performed experiments on eight datasets this way, where the underlying number of dimensions was varied from two to five. Four out of the eight datasets had 10 products, and 10 alternatives per product. The remaining four datasets had 15 products, and 5 alternatives per product. The number of customers was set to 150 in all datasets.

In Fig. 4 and Fig. 5 the resulting plots of fit vs. number of dimensions are shown. Elbows are clearly visible for each problem instance.

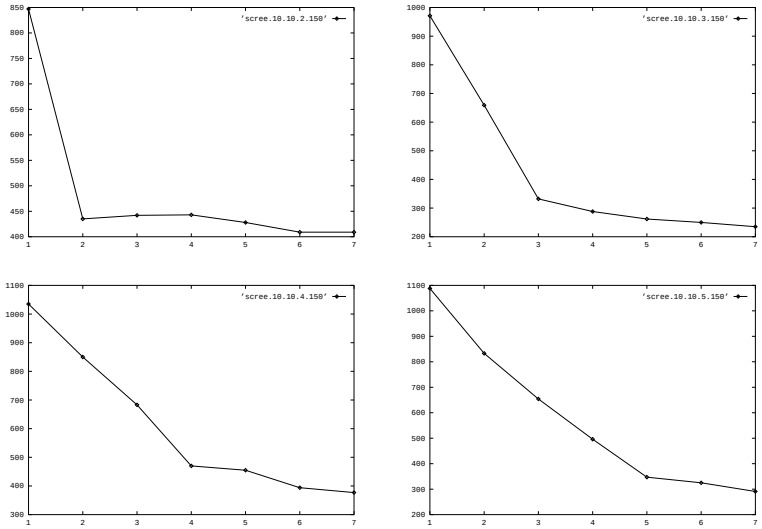

Fig. 4. The "fit vs. number of dimensions"-plots for the datasets with 10 product categories and 10 options per category.

We may conclude that the neural network is capable of discovering the dimension of the embedding space in the case of artificial data.

6.2 Results on Artificial Data Using the Product Model

In this subsection we shall describe several experiments that were performed on artificial data using the product model. We shall also draw some conclusions for the real situation. Let us now first restrict our attention to dimension $k = 2$ in the product model. In this case results can be easily visualised. We

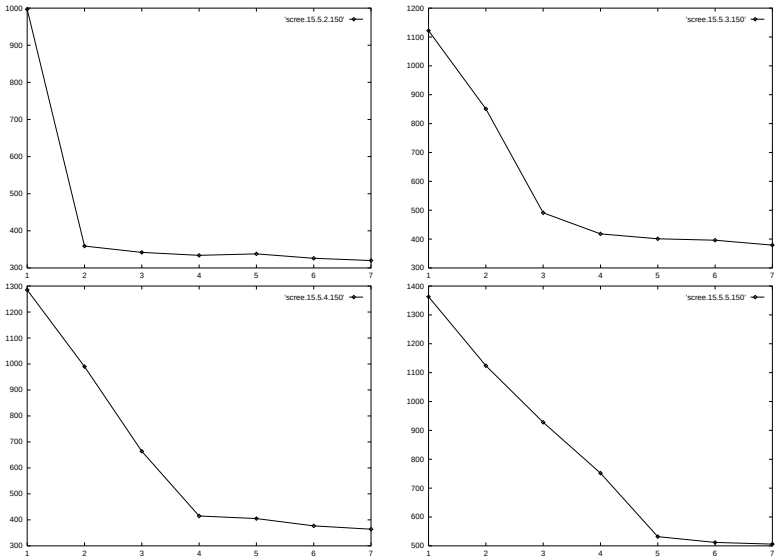

Fig. 5. The "fit vs. number of dimensions"-plots for the datasets with 15 product categories and 5 alternatives per category.

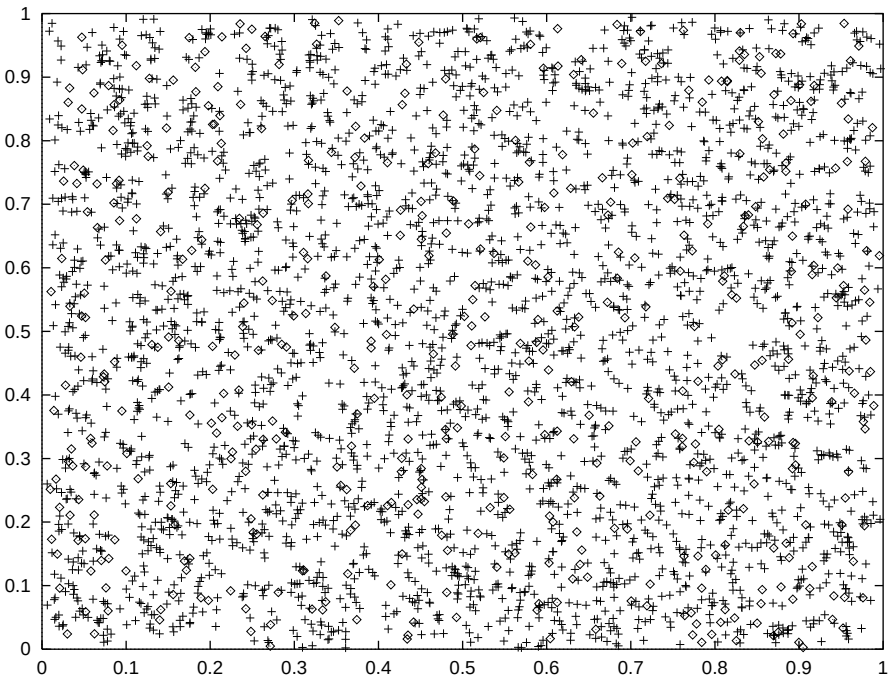

Fig. 6. Resulting customer and product coordinates of example run 1.

chose $n = 500$ customers and $m = 3,000$ products, so experiments took little time. As a good run we present example run 1: a situation where 116,638 products were sold, whereas the total error was a mere 8,928, with 7,122 "only model bought". In this case every customer bought at least about 200 products. Fig. 6 shows both customer and product coordinates; customers correspond to small squares, products to small +'s.

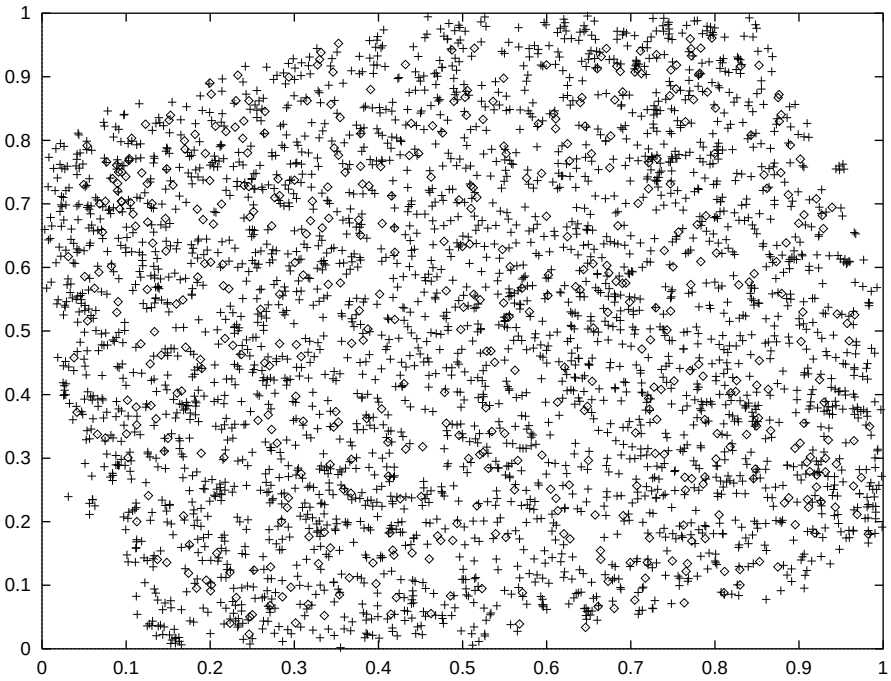

Fig. 7. Resulting embedding of example run 2.

As a second example (see Fig. 7) we consider a situation where we achieved a total error of 21,015, with 20,773 "only model bought". The total number of sales was 117,997. This example shows the importance of repeated runs—the parameters were exactly the same as in the previous example, but this time the unit square was only partially used. The result is not really bad, but if one fills the whole square, as in the previous example, a better result is possible. Note that the coordinates found are situated in a rectangular shape, suggesting that the regained coordinates have almost the same positions relative to each other as the original ones. This illustrates the fact that there are some degrees of freedom (i.e., rotations and scaling) in the behaviour of the model.

As a final example for dimension $k = 2$ we present example run 3. This was an experiment with few sales. In this case only 14,985 products were sold.

The total error was 24,094, with 20,850 "only model bought". Fig. 8 is representative for many experiments: the final error is large, whereas customers and products are not evenly distributed in space, contrary to the artificially generated ones.

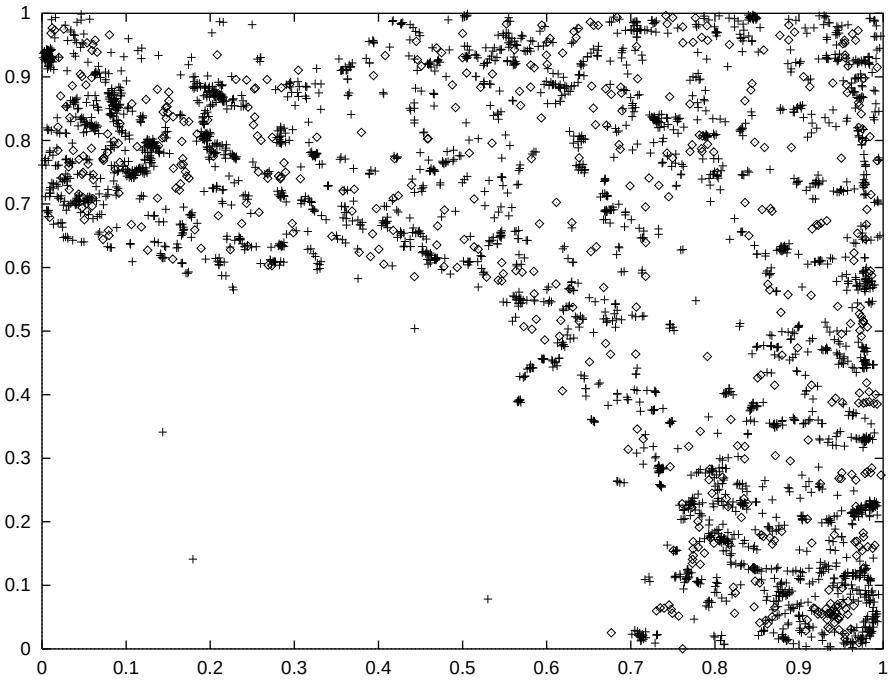

Fig. 8. Resulting embedding of example run 3.

From the experiments we concluded that the best results were obtained when customers bought very many products. Intuitively customers that buy only a few products are not representative, and from a marketing point of view, these customers are perhaps not the most interesting ones. In the sequel we therefore restrict our attention to customers buying at least 40 products.

In Table 1 we give some results in the case of artificial data for higher dimensions, where every customer bought about 50 products. The number of customers considered was near to 200. The experiments are labeled A through I. As an illustration the number of cycles is also tabulated. Here the experiments G, H and I had slightly different parameters. Some runs were almost perfect. Again we may conclude that the neural network is capable of fitting the coordinates in the case of artificial data.

In Table 2 we present some simple theoretical values. We denote by "k-wrong" a situation where the total number of products bought matches the

name	dimension	number of cycles	"both bought"	"only reality bought"	"only model bought"
A	4	2,820	6,868	493	83
B	5	2,460	12,323	1,294	267
C	6	2,400	12,348	1,768	331
D	7	2,000	13,422	1,648	345
E	8	2,060	12,072	356	207
F	9	1,480	10,956	338	91
G	10	1,680	6,695	25	12
H	10	1,760	6,115	38	17
I	10	1,320	24,426	4,259	1,042

Table 1. Results for some experiments on artificial data.

corresponding number from the sales list exactly, but precisely k products are replaced by incorrect ones. Of course, perfect is 0-wrong.

name	dimension	"both bought"	"only reality bought"	"only model bought"
naive	any	0	t	0
perfect	any	t	0	0
k-wrong	any	$t-k$	k	k

Table 2. Some theoretical values.

7 Results on Real Data

In this section we present some exploratory results from experiments on real data. Again, first the results are given for the option model and then the results for the product model. In the case of the option model the aim was to discover the most suitable dimension of the underlying space, whereas the aim in the case of the product model was merely minimization of the error, and studying the behaviour of the model.

7.1 Results on Real Data Using the Option model

After a positive result was obtained with the option model on artificial data, we considered real data. The real data were obtained from a wholesale department. The subdivision of products into product categories was performed by domain experts from the wholesale department.

From each original dataset available we selected only the customers that actually bought a product from each product category, so we did not have to

deal with the non-buying problem mentioned in Section 2. This left us with approximately 150 customers per dataset, about the same number as in the artificial datasets. The number of product categories was 11 for each dataset.

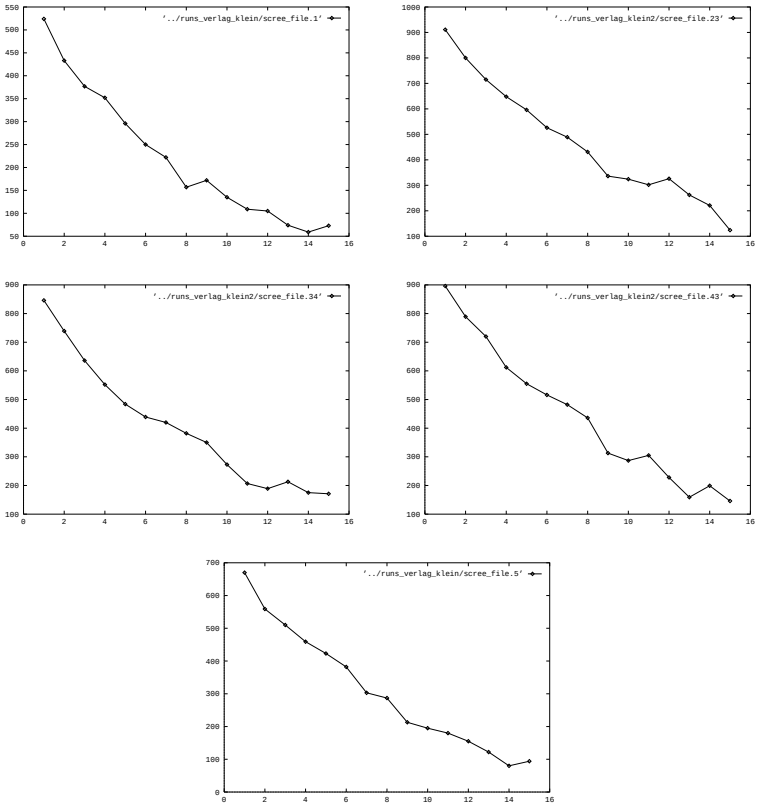

Fig. 9. The "fit vs. number of dimensions"-plots for the five real datasets — option model.

The "fit vs. number of dimensions"-plots resulting from these experiments are shown in Fig. 9. On first sight, it seems that the results of these experiments are not as good as the results with the artificial data. However, in some of the plots there is a vague elbow visible. An interpretation of the dimensions that were obtained by a domain expert could shed more light on these results. Furthermore, it may be the case that in other application areas more convincing results are obtained because customers in these other areas behave more similar to the model-customers, but this is speculative.

7.2 Results on Real Data Using the Product Model

We now turn our attention to the product model. As a running example we use a real sales list consisting of $n = 1,368$ customers, who bought 32,523 products. The number of different products was $m = 10,381$. We restricted our attention to customers buying at least 40 products. These $n = 228$ customers (17% of the total number of customers) bought $t = 11,769$ products (36% of the total number of products sold). As this example will show, reasonable results can be obtained.

In Table 3 we show results from interesting runs using real data. Note that "both bought" and "only reality bought" always add to $t = 11,769$. The error is a combination of the columns "only reality bought" and "only model bought". As an illustration two experiments, X and Y, in dimension 9 have

name	dimension	number of cycles	'both bought'	'only reality bought'	'only model bought'
X	9	2,340	6,468	5,301	1,211
Y	9	1,360	5,512	6,257	668
A	10	1,820	7,072	4,697	1,138
B	10	1,440	6,010	5,759	499
C	10	1,760	7,990	3,779	5,700

Table 3. Results for some experiments.

been added. The results arise from slightly different settings of parameters, and from repeated experiments with the same parameters. Experiment A shows a relatively good result. An average customer buys 52 products and the model correctly buys 31 of these, where instead of the remaining 21 it buys 5 incorrect products. For experiment B this last number is lowered to 2, but only 26 products are properly handled on average. In experiment C we see that on average even 35 products out of 52 are correctly predicted by the model; raising "both bought" seems only possible at the cost of raising "only model bought" too.

As a visualisation of experiment A we plot the error against the number of cycles. In Fig. 10 we show the "relative error": the sum of "only reality bought" and "only model bought" divided by "both bought".

From the experiments we noticed that the best results were achieved when using the Euclidean distance d, defined by $d(x, y) = \sqrt{\sum_{\ell=1}^{k}(x_\ell - y_\ell)^2}$ for vectors $x = (x_1, \ldots, x_k)$ and $y = (y_1, \ldots, y_k)$ in k-space. We also experimented with Manhattan-like distances (taking the sum of the absolute values of $x_\ell - y_\ell$), but results were not very promising in that case: the errors obtained were much larger.

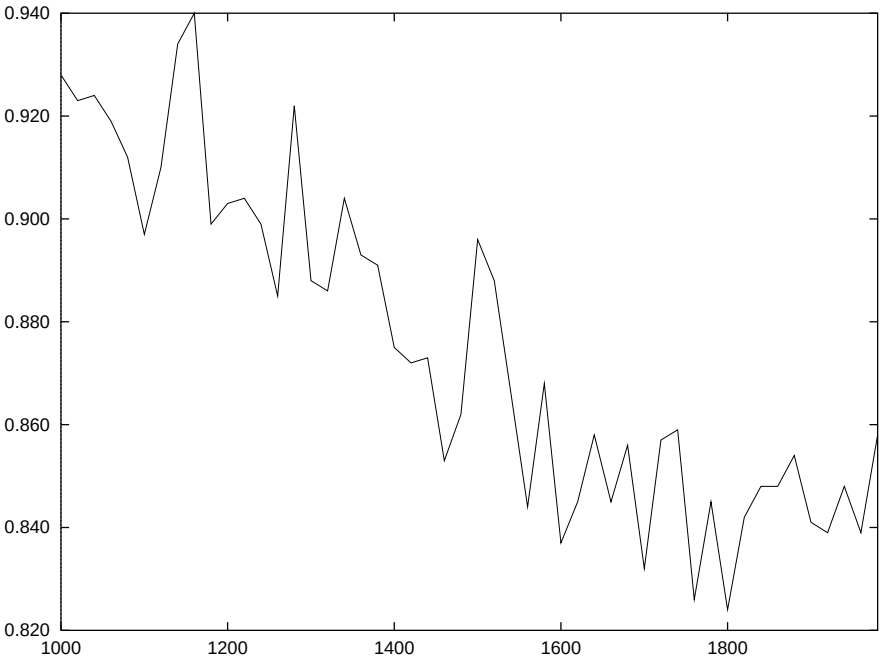

Fig. 10. Relative error vs. number of cycles.

As in the case of the option model, results are not as good as those for artificial data. We feel however that the results are promising, and may even be better for certain special application areas.

8 Summary and Conclusions

In this paper we described two models, the option model and the product model, to understand and analyse customer choices. The theoretical complexity of special cases of these models was addressed and the corresponding decision problems were shown to be NP-complete.

Using techniques from neural networks, we were able to obtain good results for artificial data using both the option model and the product model. For real data, the results were less convincing, but in the case of the product model where customers buy lots of products, the results were still reasonable.

At the moment it is not possible to conclude whether or not these models reflect real life customer choices. Of course, it would be unrealistic to expect customers to behave exactly as in a model, but we do hope to discover underlying trends. Feedback from domain experts may reveal the possible meaning of the dimensions.

Similar techniques have been used in the following situation. If one tries to understand customer behaviour in the case of different shops or products, 2-

dimensional plots showing their relative positions might be useful. These plots can be generated quite easily with the methods described above. In fact, once a notion of distance between shops/products has been introduced, the neural networks quickly discover reasonable embeddings in 2-space. The distance function is usually taken to be some sort of Hamming-distance, counting the number of absolute differences between vectors. In the shop case the vectors might consist of the week sales; in the product case the distances can be based on the number of times the products have been bought together. Note that the resulting networks resemble those using gravity from [14].

Finally, we would like to discuss the practical use of the methods discussed so far. One should not expect the answers to be precise and rigid. On the contrary, they rather give insights and ideas. One might think, for instance, that customers that are near to one another in k-space show similar behaviour. This is not necessarily true, but in many situations these customers do resemble each other. This property can be used in marketing situations. Not only is it possible to reveal interesting dimensions, also interesting customers or clusters of customers can be detected. Due to the randomness of the algorithms involved repeated runs might give somewhat or even totally different results. This does not mean that the methods are not sound, but it shows the difficulty of the problem at hand, reflected in the enormous number of solutions of acceptable quality.

References

1. Anderson, J.A., Rosenfeld, E. (Eds.) (1988): Neurocomputing: Foundations of Research. MIT Press, Cambridge
2. Bishop, C.M. (1995): Neural Networks for Pattern Recognition. Clarendon-Press, Oxford
3. Davison, M.L. (1983): Multidimensional Scaling. John Wiley and Sons, New York
4. Fayyad, U., Uthurusamy, R. (1996): Data Mining and Knowledge Discovery in Databases. Communications of the ACM 39, 24–27
5. Garey, M.R., Johnson, D.S. (1979): Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York
6. Grossberg, S. (1976): Adaptive Pattern Classifications and Universal Recording, I: Parallel Development and Coding of Neural Feature Detectors. Biological Cybernetics 23, 121–134
7. Grossberg, S. (1980): How Does a Brain Build a Cognitive Code? Psychological Review 87, 1–51; reprinted in [1]
8. Haykin, S. (1999): Neural Networks: A Comprehensive Foundation, 2nd edition. Prentice Hall, Upper Saddle River, New Jersey
9. Hertz, J., Krogh, A., Palmer, R.G. (1991): Introduction to the Theory of Neural Computation. Addison-Wesley, Reading, Massachusetts
10. Kosters, W.A., La Poutré, H., Wezel, M.C. van (1997): Understanding Customer Choice Processes Using Neural Networks. In: Arner Jr, H.F. (Ed.): Proceedings of the First International Conference on the Practical Application of Knowledge Discovery and Data Mining (PADD'97), London, 167–178

11. Kotler, P. (1999): Marketing Management: Analysis, Planning, Implementation and Control, 9th edition. Prentice Hall, Upper Saddle River, New Jersey
12. Krzanowski, W.J. (1988): Principles of Multivariate Analysis. Oxford Statistical Science Series, Oxford University Press, Oxford
13. Leeuw, J. de, Heiser, W. (1982): Theory of Multidimensional Scaling. In: Krishnaiah, P.R., Kanal, L.N. (Eds.): Handbook of Statistics 2: Classification, Pattern Recognition and Reduction of Dimensionality. North-Holland, Amsterdam, 285–316
14. Oyang, Y.-J., Chen, C.-Y., Yang, T.-W. (2001): A Study on the Hierarchical Data Clustering Algorithm Based on Gravity Theory. In: De Raedt, L., Siebes, A. (Eds.): Proceedings PKDD 2001 (Principles of Data Mining and Knowledge Discovery), Lecture Notes in Artificial Intelligence 2168, Springer, Berlin Heidelberg New York, 350–361
15. Wezel, M.C. van, Kok, J.N., Sere, K. (1996): Determining the Number of Dimensions Underlying Customer-Choices with a Competitive Neural Network. In: Proceedings of the IEEE International Conference on Neural Networks (ICNN'96), Washington D.C., 484–490