Journal of Business Economics and Management
ISSN 1611-1699 print / ISSN 2029-4433 online
2013 Volume 14(5): 923–939
doi:10.3846/16111699.2012.749807

CUSTOMER SEGMENTATION IN PRIVATE BANKING SECTOR USING MACHINE LEARNING TECHNIQUES

Ion Smeureanu[1], Gheorghe Ruxanda[2], Laura Maria Badea[3]

Faculty Cybernetics, Statistics and Economic Informatics,
The Bucharest University of Economic Studies,
6 Piata Romana, 1st district, 010374 Bucharest, Romania
E-mails: [1]smeurean@ase.ro; [2]gheorghe.ruxanda@csie.ase.ro;
[3]laura.maria.badea@gmail.com (corresponding author)

Received 25 September 2012; accepted 12 November 2012

Abstract. Machine learning techniques have proven good performance in classification matters of all kinds: medical diagnosis, character recognition, credit default and fraud prediction, and also foreign exchange market prognosis. Customer segmentation in private banking sector is an important step for profitable business development, enabling financial institutions to address their products and services to homogeneous classes of customers. This paper approaches two of the most popular machine learning techniques, Neural Networks and Support Vector Machines, and describes how each of these perform in a segmentation process.

Keywords: machine learning, neural networks, support vector machines, customer segmentation, private banking.

Reference to this paper should be made as follows: Smeureanu, I.; Ruxanda, G.; Badea, L. M. 2013. Customer segmentation in private banking sector using machine learning techniques, *Journal of Business Economics and Management* 14(5): 923–939.

JEL Classification: C38, C45, C63.

1. Introduction

Profitable business development strategies always begin with good customer segmentation. This enables economic entities to identify specific characteristics of their products and services demanders, and elaborate efficient business plans. In the banking sector the process of customer segmentation has become a useful tool in gaining more customers, but also in extracting a higher value from the existing ones.

Market segmentation is a set of concepts and models that guides management thinking and leads to new profitable product/service offerings. The general assumptions are that customers with resembling characteristics have similar needs. Therefore, there is high interest in identifying homogeneous groups of existing customers or potential ones whose demand for distinct configured products sustains a promising market opportunity.

Experience has proven that an effective segmentation process eventually brings high benefits. However, before this happens, there are some important aspects that pre-condition the achievement of accurate models and often these relate to: variable selection and statistical methods chosen to perform the segmentation.

2. Binary classification methods

Data classification becomes each and every day an important problem, as the amount of information that is generated and needs to be classified becomes larger. Traditional statistical methods such as Discriminant Analysis and Logistic Models have been replaced in many cases by non-parametric techniques like Neural Networks and Support Vector Machines (henceforth SVMs). Thus, in recent years, machine learning has become one of the most promising alternatives to solve binary classification problems. SVMs have already proven good results in fields like medical diagnosis (Noble 2006), character recognition (Ahmad *et al.* 2004), and credit scoring (Auria, Moro 2008).

The present paper analyzes the predictive power of a private-banking segmentation model using both Neural Networks and SVMs.

2.1. Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational methods inspired by the way in which the human brain functions. They are non-linear statistical data modeling techniques in which interconnected elements process the information simultaneously, having the ability to adapt and learn from past examples. Artificial Neural Networks have application in various fields such as: *pattern recognition*; *medical diagnosis* (Ultsch *et al.* (1995) used the capacity of neural network to diagnose acidosis diseases; Zhou *et al.* (2001) proposed an Artificial Neural Network ensemble in the process of lung cancer diagnosis; Kiyan and Yildirim (2003) used ANNs for breast cancer diagnosis); *credit risk models* (Amir (2001) developed a Neural Network bankruptcy prediction model; Baesens *et al.* (2003) used Neural Network rule extraction for credit risk evaluation); *market prices changes* (Yoon and Swales (1993) analyzed the forecasting power of Neural Networks for stock prices) etc.

First introduced by McCulloch and Pitts (1943), Neural Networks have passed through different stages over the following decades. One important development was made by Rosenblatt (1958) on alphanumerical character recognition when he presented the perceptron. Later, in their paper "Perceptron", Minsky and Papert (1969) managed to put to a doubt the modeling capacity of Neural Networks in solving linearly non-separable problems. No sooner than 15 years later, have Neural Networks started to regain strength through a number of papers outlining the capacity of multi-layered networks to solve non-linear problems.

Multilayer perceptrons (henceforth also referred to as MLPs) have three types of layers, namely one *input layer*, one or more *hidden layers*, and one *output layer*. The input layer handles the initial data features, while the output layer depends on the type of answer of the network. Knerr *et al.* (1992) have shown that one hidden layer usually

manages to solve very well classification problems, but in extreme cases the number of hidden layers can go up to three (a higher number of hidden layers can lead to over-fitted models). However, there is no recipe in respect with the number of hidden neurons and most of the times, this is selected based on different trials and error comparisons.

Depending on the architecture of the MLPs, we distinguish feed-forward (Fig. 1) and feed-back networks (recurrent). The former permits the propagation of information only in one direction and never between neurons from the same layer, while the latter are more complex and signal can be processed several times within the same layer without moving to the next layer.

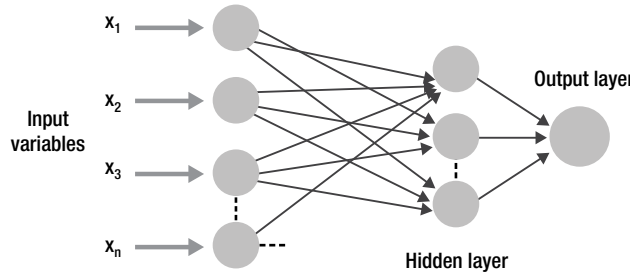

Fig. 1. Feed-forward network

Back-propagation

The most popular and comprehensible type of feed-forward network is the "back-propagation" which was first proposed by Rumelhart *et al.* (1986). The most common back-propagation neural network uses the *steepest descent* (*gradient descent*) algorithm. According to this method, weights are adjusted in the direction corresponding to the negative gradient of the error surface. Nevertheless, the gradient does not indicate the global minimum, inducing the risk of getting blocked in a local minimum. This direction can be guided by two adjustable parameters such as *learning rate* and *momentum rate*. The first one is a parameter that controls the step size when weights are iteratively adjusted. Small *learning rates* generate slow convergence but impose some computational costs, while high values converge faster to the real solution, but might also overstep it or go into the wrong direction. The latter parameter, *momentum rate*, refers to the fact that previous changes in the weights should impact the current direction of movement in the weight space. However, the optimal values for these parameters depend on the type of problem one wants to solve and are obtained by making several tests.

Usually, the training process of a "back-propagation" algorithm involves a number of steps aimed to decrease the validation sample error.

Initially, one has to define the architecture of the neural network and select the appropriate input variables. Afterwards, the weights (w) and threshold values (θ) are initialized on a random basis. The training data and the output matrix are afterwards introduced (x and T respectively) in the model.

Then, for every neural node, one has to perform the output values. This is done both for the hidden layer (resulting thus the output vector *H*) and output layer (with output vector *Y*):

$$u_k = \sum w_{ki} x_i - \theta_k \,, \tag{1}$$

$$H_k = f(u_k) \tag{2}$$

for the hidden layer and:

$$u_j = \sum w_{jk} H_k - \theta_j \,, \tag{3}$$

$$Y_j = f(u_j) \tag{4}$$

for the output layer,

where x_i is the value of input variable i, w_{ki} is the weight giving the intensity of the signal induced by input variable i to the hidden neuron k, θ_k is the threshold value, u_k is the input for hidden unit k, H_k is the output of the hidden unit k, $f(u_k)$ is the activation function for the hidden unit k, w_{jk} is the link from hidden neuron k to output layer j, θ_j is the threshold value, u_j is the input for output unit j, Y_j is the output for the final layer, and $f(u_j)$ is the activation function for the output unit j.

Based on the outputs, the sum of squares is calculated:

$$SOS = \sum (Y_j - T_j)^2 \,, \tag{5}$$

where SOS is the statistic sum of squares, performed as a deviation of the estimated output Y_j from the real output value T_j.

The distances between the output and hidden layer are computed as follows:

$$\delta_j = (T_j - Y_j) \times f'(u_j) \tag{6}$$

and

$$\delta_k = (\sum\nolimits_j \delta_j w_{jk}) \times f'(u_k) \,, \tag{7}$$

where δ_j is the distance reached in the output layer j, and δ_k is the distance determined for the hidden layer k.

Then, the weights (w) and threshold values (θ) are modified accordingly in each type of layer:

$$\Delta w_{jk}(n) = \eta \delta_j H_k + \alpha \Delta w_{jk}(n-1) \,, \tag{8}$$

$$\Delta \theta_j(n) = -\eta \delta_j + \alpha \Delta \theta_j(n-1) \tag{9}$$

for the output layer, and:

$$\Delta w_{ki}(n) = \eta \delta_k x_i + \alpha \Delta w_{ki}(n-1) \,, \tag{10}$$

$$\Delta \theta_k(n) = -\eta \delta_k + \alpha \Delta \theta_k(n-1) \tag{11}$$

for the hidden layer,

where $\Delta w_{jk}(n)$ is the modification of the weight from hidden neuron k to output neuron j for the current step n, $\Delta w_{jk}(n-1)$ is the modification from the previous step $n-1$, $\Delta\theta_j(n)$ is the current modification for the threshold value, $\Delta\theta_j(n-1)$ is the previous modification of the threshold, α is the momentum coefficient according to which past modifications in the weights should impact the current evolution of the weights, and η is the learning rate which controls the size of the step when weights are iteratively adjusted. Analogous, equations (10) and (11) describe the adjustments for the weights used in the hidden layers.

Then, new w and θ are computed for the output layer:

$$w_{jk}(n) = w_{jk}(n-1) + \Delta w_{jk}, \tag{12}$$

$$\theta_j(n) = \theta_j(n-1) + \Delta\theta_j \tag{13}$$

and for the hidden layer:

$$w_{ki}(n) = w_{ki}(n-1) + \Delta w_{ki}, \tag{14}$$

$$\theta_k(n) = \theta_k(n-1) + \Delta\theta_k \ , \tag{15}$$

where $w_{jk}(n)$ is the adjusted weight for step n, giving the intensity that the value of hidden neuron k has on output neuron j and which is determined based on the value of the weight from the previous step ($w_{jk}(n-1)$) and the adjustment calculated in equation (8). The new threshold value, $\theta_j(n)$, is performed on a similar basis, adding the modification computed in equation (9) to the threshold value from the previous step, $\theta_j(n-1)$. For the hidden layer, the process is likewise.

This process, from equations (1) to (15), is repeated until the sought criterion in respect with error is met.

2.2. Support Vector machine

Support Vector machine (SVM) represents a concept, originating from the field of statistics and computer science, that describes a set of methods for data analysis and pattern recognition. The original idea and initial development of the method belongs to Vladimir Vapnik, and was introduced in 1963, and developed later in the 90's by Vapnik and his co-workers (Boser *et al.* 1992; Cortes, Vapnik 1995; Vapnik 1995). The standard SVM uses an input dataset and for each sample performs a binary classification. This behavior puts SVM method in the non-probabilistic binary linear classifier group. The initiator of the SVM model, Vapnik, proved its performance on a number of problems, most notable at that time being handwriting recognition.

The prime aspects related to SVMs are presented by Ayodele (2010: 25)[1]*: attributes* which are predictor variables; *features* that represent transformed attributes that define

[1] Taiwo Oladipupo Ayodele, from University of Portsmouth from United Kingdom presented this in chapter 3 "Types of machine Learning Algorithms", from the book edited by Yagang Zhang "New Advances in Machine Learning", 2010.

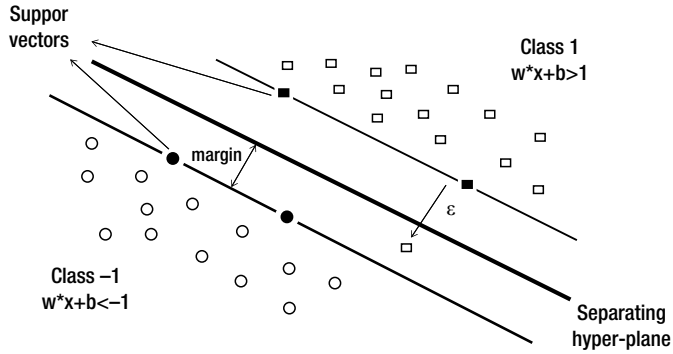

Fig. 2. Support Vector representation

the hyper-plane; *vectors* which are sets of features that describe one case; *Support Vectors* representing the vectors near the hyper-plane that have the role of bounding the hyper-plane (Fig. 2).

A Support Vector machine is a mathematical algorithm that maximizes the function which defines a particular set of data. As Noble (2006) claims, the main idea behind the SVM, relates to four basic concepts which will be further used in defining the model:

- *Separating hyper-plane*, a line which acts as a separator between the two classes. There can be more than one separating hyper-plane;
- *Maximum margin hyper-plane*, the hyper-plane that provides the highest distance between the separating hyper-plane and the nearest expression vectors;
- *Soft margin*, that allows wrongly classified cases by the separating hyper-plane;
- *Kernel function*, which transposes the data from a low-dimensional space into another one of a higher dimension.

However, projecting the data into very high-dimensional space, just so that the soft margin decreases, is not always suitable due to the "*curse of dimensionality*"; i.e. when the number of variables increases, the number of solutions rises at a higher speed and this makes the classification more difficult and the over-fitting phenomenon appears. When the model is over-fitted it is very likely that it will not perform well on an "out-of-sample" data.

Mathematical model

If a new object (individual) j should be classified into one of two classes (class1 or class2) following a linear SVM score, it would look like this:

$$s_j = x_j^T w + b, \tag{16}$$

where $x_j = (x_{j1}, x_{j2}, ..., x_{jd})$ is a vector with d variables; x_{jk} is the value of variable k for object j, $k = 1, ..., d$; w is the vector containing the weights of the d variables, and b is a constant.

In order to develop such a model, a SVM learns on a training sample the values of parameters *w* and *b*. Geometrically, this means identifying a hyper-plane that performs as a separator between the two classes based on a certain criterion that is induced by margin maximization. This margin represents the distance between the hyper-planes which limit each class (in a perfect model no object is misclassified). By maximizing the margin, we identify the function that best separates objects in class1 from objects in class2. These two margin boundaries are: $x^T w + b = 1$ and $x^T w + b = -1$, and thus, the margin becomes $2/\|w\|$, where $\|w\|$ is the norm of vector *w*.

When not all objects are correctly classified, the margin is said to be *soft*. Consider that ε_i is a variable for misclassifications ($\varepsilon_i \geq 0$). When $\varepsilon_i = 0$, then all objects are correctly classified. Otherwise, it means there are misclassifications, which creates the need of finding a criterion for minimizing these training errors and further determine the parameters *w* and *b*.

Using the constraint of having no object laying within the margin, except some classification errors, the following condition is defined:

$$y_i(x_i^T w + b) \geq 1 - \varepsilon_i, \; i = 1,...,n, \tag{17}$$

where y_i is the variable indicating the class of an object (–1 or 1).

Following, this becomes an optimization problem for the calculation of *w* and *b*.

$$\min_w \frac{1}{2}\|w\|^2 + C\sum_i^n \varepsilon_i \tag{18}$$

with constraints (17) and $\varepsilon_i \geq 0$.

Relation (18) first maximizes the margin $[2/\|w\|]$ by minimizing $\frac{1}{2}\|w\|^2$, where $\|w\|^2$ comes from the second term representing the sum of misclassification errors $[\varepsilon_i/\|w\|]$ multiplied with parameter *C*. This leads to a convex quadratic problem, a maximization of the margin width with minimizing errors.

Parameter *C* weights classification errors and induces the generalization capacity of a SVM. The higher is *C*, the higher is the weight given to misclassifications, the lower the generalization of the machine is. Low generalization means that the machine may work well on the training set, but perform bad on new samples.

Bad generalization may be a result of over-fitting on the training sample, for example, in the case that this sample shows some untypical data structure. In order to reduce the risk of over-fitting a SVM on the training sample, one has to choose a low *C*. The smaller *C* is, the wider the margin is, and thus more and larger in-sample classification errors are permitted.

The above optimization problem can be solved using Lagrange function:

$$L(w,b,\varepsilon,\alpha,v) = \frac{1}{2}w^T w + C\sum_i^n \varepsilon_i + \sum_i^n \alpha_i\{y_i(w^T x_i + b) - 1 + \varepsilon_i\} - \sum_{i=1}^n v_i\varepsilon_i, \tag{19}$$

where $\alpha_i \geq 0$ are the Lagrange multipliers for the constraint (17) and $v_i \geq 0$ are Lagrange multipliers for constraint $\varepsilon_i \geq 0$.

By applying Kuhn-Tucker Theorem, the solution to this problem is the saddle-point of the Lagrangian, minimized in respect with *w*, *b*, and ε_i, and at the same time maximized in respect with α and *v*. This can be reduced to a convex quadratic programming problem in α_i:

$$w = \sum_{i=1}^{n} y_i \alpha_i x_i, \tag{20}$$

$$b = \frac{1}{2}(x_{+1}^T + x_{-1}^T) \times w, \tag{21}$$

where x_{+1}^T and x_{-1}^T are any two Support Vectors belonging to two different classes, which lie on the margin boundaries.

Those cases where $\alpha_i \neq 0$, are called *Support Vectors* and they are relevant for determining *w*. These are on the margin boundaries or within the margin for non-perfectly separable data. As Downs *et al*. (2001) say, the time taken for a Support Vector classifier to compute the class of a new pattern is proportional to the number of Support Vectors. Therefore, if that number is large, classification speed is slow.

Introducing (20) into (16), we obtain the score s_j as a relation of the scalar product of the variables of object *j* to be classified, and the variables of the Support Vectors in the training sample, of α_i, and of y_i. By comparing s_j with a benchmark value, we are able to estimate if object *j* has to be classified in class1 or class2:

$$s_j = \sum_{i=1}^{n} y_i \alpha_i \left\langle x_i, x_j \right\rangle + b. \tag{22}$$

Kernel function

When the SVM is non-linear, the score of an object is determined by replacing the scalar product of variables with a kernel function $K(x_i, x_j)$. Thus, equation (22) becomes:

$$s_j = \sum_{i=1}^{n} y_i \alpha_i K(x_i, x_j) + b. \tag{23}$$

The performance of a SVM is highly related to the kernel function (Amari and Wu 1999). Nevertheless, there is no theory regarding the method selection of the kernel and in most cases this is done by making several trials and error comparisons. Some of the most popular types of kernel functions are: polynomial, sigmoid and RBF (radial basis function).

Kernels are symmetric, semi-positive definite functions which satisfy Mercer Theorem[2]. If this theorem is satisfied, this ensures that there exists a (possibly) non-linear map Φ from the input space into some feature space, such that its inner product equals the kernel. The non-linear transformation Φ is only implicitly defined through the use of a kernel, since it only appears as an inner product.

[2] Let *K*: $R^n \times R^n \to R$ be given. Then for the *K* to be a valid (Mercer) kernel, it is necessary and sufficient that for any $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$, ($m < \infty$), the corresponding kernel matrix is symmetric positive semi-define.

$$K(x_i, x_j) = \left\langle \Phi(x_i), \Phi(x_j) \right\rangle. \tag{24}$$

The classification problem is solved this way: the input space is transformed by Φ into a feature space of a higher dimension, where it is easier to find a separating hyperplane. Thus, the Kernel can solve the problem that data are non-linearly separable, by implicitly mapping them into a feature space, in which the linear threshold can be used. Using a kernel is equivalent to solving a linear SVM in a new higher-dimensional feature space. The non-linear SVM score is therefore a linear combination, but with new variables, which are derived through a kernel transformation of the prior variables. The score function does not have a compact functional form, depending on a transformation of the variables, which we do not know, since it is only implicitly defined. The solution of the constrained optimization problem for non-linear SVM is given by:

$$w = \sum_{i=1}^{n} y_i \alpha_i \Phi(x_i), \tag{25}$$

$$b = -\frac{1}{2}\left(\sum_{i=1}^{n} y_i \alpha_i K(x_i, x_{+1}) + \sum_{i=1}^{n} y_i \alpha_i K(x_i, x_{-1})\right). \tag{26}$$

According to (23) and (26), we do not need to know the form of the function Φ for the score calculation. For the calculation of the score (22), the input variables are used as a product. Thus, only the kernel function is needed in equation (26). As a consequence, Φ and *w* are not required for the solution of a non-linear SVM.

The Gaussian kernel is by far one of the most versatile Kernels. It is a radial basis function kernel, and is the preferred Kernel when we don't know much about the data we are trying to model:

$$K(x_i, x_j) = e^{-\left(\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)}. \tag{27}$$

The adjustable parameter sigma plays a major role in the performance of the kernel, and should be carefully adjusted to the problem at hand. If it is overestimated, the exponential will act almost linearly and the higher-dimensional projection will start to lose its non-linear power. On the other hand, if underestimated, the function will lack regularization and the decision boundary will be highly sensitive to noise in training data.

3. Models comparison

A SVM model using a sigmoid kernel function[3] is the same as a two-layer, perceptron neural network. Support Vector Machines are related to classical multilayer perceptron Neural Networks. Using a kernel function, SVM's are an alternative training method for polynomial, radial basis function and multi-layer perceptron classifiers in which the weights of the network are found by solving a quadratic programming problem with linear constraints, rather than by solving a non-convex, unconstrained minimization problem, as in standard neural network training.

[3] $K(x_i, x_j) = \tanh(\alpha x_i^T x_j + r)$.

Nevertheless, a significant advantage of SVMs is that while ANNs can suffer from multiple local minima, the solution to a SVM is global and unique. Other advantages of SVMs are that they have a simple geometric interpretation and give a sparse solution. Unlike ANNs, the computational complexity of SVMs does not depend on the dimensionality of the input space. ANNs use empirical risk minimization, whilst SVMs use structural risk minimization. The reason for which SVMs often outperform ANNs in practice is that they deal with the biggest problem of Neural Networks, i.e. SVMs are less prone to over-fitting. SVMs offer good results for out-of-sample generalizations, for properly set *C* and *gamma* parameters (when using Gaussian kernels). This makes them suitable to training sets that are biased at some degree.

As disadvantages, Auria and Moro (2008) mention their lack of transparency of the results, because scores for particular observations cannot be easily obtained. SVMs do not offer scores in the form of simple parametric functions, because the contribution of each parameter is not constant for all observation points. Other disadvantages identified when comparing SVMs with Neural Networks point to SVMs being only binary classifiers. This limits their usage, and imposes the creation of more complex models to solve multi-class problems for a set of observations.

4. Data sample and variables selection

The database consists of 2,783 observations representing active cardholders at an important commercial bank from Romania. Based on the transactions performed over the previous six months as related to 30th June 2011, we want to identify those customers that bring benefit to the bank through certain habits, so as to target them with other specific products (deposits, credit cards etc.). This means, we are interested in distinguishing the "*affluent"* customers from those with "*mass"* characteristics based on some specific behavior. For the classification model an a-posteriori status must be assigned to all customers and the used indicator was the average monthly sum of transactions. Those clients overcoming the 4,500 RON threshold were classified as "*affluent*", whereas the others are "*mass*" clients. On this basis, 336 individuals were labeled "*affluent*" and the remaining ones, "*mass*" customers.

Variable selection is a very important process when performing segmentation. Practically, different combinations of these characteristics will stay at the basis of the classification process and thus it is highly important to have relevant information. Frank, Massy and Wind (1972) defined two categories of variables, i.e. *general descriptive customer characteristics* and *characteristics related to customer behavior*, while Kotler (1998) presented four distinct classes of variables: *geographical*, *demographical*, *psycho-graphical* and *behavioral*.

For our data sample, information regarding customer transaction behavior within the past six months was processed on the basis of the available transaction history. The information regarding general characteristics of the customers was however very limited and eventually only one variable of this type was selected[4].

[4] Customer domicile (capital/province).

Since neither Neural Networks nor SVMs impose any assumptions and restrictions regarding the quality of the data, variable selection was performed on statistical basis but also on expert judgment. Based on this, 31 variables (out of which 30 continuous and one categorical) resulted, and these are presented in Table 1.

Table 1. List of predictors

	Variables
1	maximum transaction value within a month
2	maximum transaction value within one single day
3	maximum no. of transactions within one single day over the past six months
4	maximum no. of transactions within one month
5	maximum transaction value at other than ATM
6	maximum transaction value
7	average monthly no. of transactions
8	maximum transaction value at ATM
9	proportion of POS transactions within overall transactions by no.
10	proportion of ATM transactions within overall transactions by no.
11	proportion of transactions made in other county than the domicile within overall transactions by no.
12	proportion of transactions in other county than domicile within total transactions by no.
13	minimum transaction value at other than ATM
14	no. of cards
15	proportion of foreign countries transactions within overall transactions by no.
16	minimum transaction value at ATM
17	sum of transactions performed in hotels
18	no. of transactions in hotels
19	customer domicile (capital/province) – *binary categorical variable*
20	sum of transactions at jewelry/watches/expensive glass stores
21	no. of transactions at jewelry/watches/expensive glass stores
22	sum of *e*-banking transactions
23	proportion of e-banking transactions within overall transactions by no.
24	no. of transactions in book stores
25	no. of transactions at expensive restaurants
26	no. of transactions for non-low-cost airlines flights.
27	proportion of Bucharest transactions within total transactions by no.
28	proportion of other types of transactions within overall transactions by no.
29	minimum transaction value
30	proportion of ATM transactions within overall transactions by value
31	proportion of Retailers transactions within overall transactions by no.

Fig. 3 presents a quadratic 3D representation of the customer status ("*affluent*"/"*mass*") and two predictive variables, i.e. "*Number of cards"* and "*Average monthly number of transactions"*. It is visible that the higher the number of cards and also the average monthly number of transactions are, the higher the chances are that the customer is "*affluent*" type.

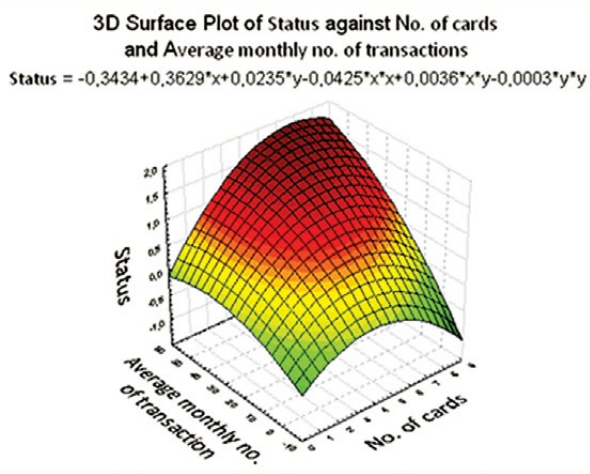

Fig. 3. Status against two predictors – 3D representation

5. Model development and results

5.1. Neural Networks

In the process of training a neural network a validation sample is required. This is because in order to avoid an over-fitted model which would eventually lead to a small error on development sample, the progress can also be checked against an independent set (validation sample). This will make the training process stop when the validation error stops decreasing. Thus, for building the neural network model, the data was split as follows: 60% for the training sample; 20% for the validation set; 20% for an out-of-sample testing process.

For the development process it was selected a classification Multilayer Perceptron using a back-propagation algorithm. This aims at reaching the minimum of the error function (*sum of squares*) in weight space, by using the method of *gradient descent*. This imposes the need of an activation function and we further chose the *logistic function* for both, hidden and output layers. After several tests on the *momentum coefficient* and *learning rate* a final value of 0.1 was assigned to both parameters because these two brought the best results in terms of detection rates on the test sample.

Because Neural Networks with one hidden layer perform well in most of the cases, for this customer segmentation research we selected one hidden layer as well. For this model we tested the performance evolution, as the number of hidden neurons varied

between 8[5] and 32 (the latter number represents the number of input neurons). The number of cycles per training process was set to a maximum of 200, and for each trial with different number of hidden neurons, 5 networks were trained, in the end resulting 125 networks. The first ten best results in terms of detection rates are summarized in Table 2.

Table 2. Trials for MLP selection

Network name	Training perf. %	Validation perf. %	Test perf. %	Training algorithm	Error function	Hidden activation	Output activation
MLP 32-8-2	93.297	91.007	93.885	GD[6] 191	SOS[7]	Logistic	Logistic
MLP 32-8-2	93.058	91.007	93.885	GD 193	SOS	Logistic	Logistic
MLP 32-9-2	92.879	90.647	93.705	GD 177	SOS	Logistic	Logistic
MLP 32-10-2	92.819	90.647	93.345	GD 171	SOS	Logistic	Logistic
MLP 32-8-2	92.280	90.288	92.806	GD 114	SOS	Logistic	Logistic
MLP 32-12-2	92.220	90.288	92.806	GD 119	SOS	Logistic	Logistic
MLP 32-12-2	92.160	90.288	92.806	GD 127	SOS	Logistic	Logistic
MLP 32-13-2	92.160	90.288	92.806	GD 125	SOS	Logistic	Logistic
MLP 32-9-2	92.220	90.288	92.626	GD 123	SOS	Logistic	Logistic
MLP 32-19-2	92.160	90.288	92.626	GD 145	SOS	Logistic	Logistic

The model classification was done by considering the best results on the test set ("out of sample" dataset), then on the validation set and finally on the training set, in this particular order. The best results were obtained after 191 cycles in case of MLP 32-8-2 with 8 hidden neurons, with a total detection rate on the test sample of 93.885%.

However, the detection rate for "*affluent"* customers is rather low, only 41% of the "*affluent*" customers being detected by the model (see Table 3).

Table 3. Confusion matrix for Test sample using MLP 32-8-2

		Predicted	
		mass	*affluent*
Observed	*mass*	**100%**	0%
	affluent	59%	**41%**

[5] Antkowiak (2006) proposes the following method when selecting the number of hidden neurons: $N_h = \sqrt{N_i N_o}$, where N_h is the no. of hidden neurons, N_i is the number of input neurons, and N_o is the number of output neurons. Based on the configuration of the available data, the number of hidden neurons determined after the above formula would be 8.

[6] Gradient descent.

[7] Sum of Squares.

5.2. SVM

The initial data sample was randomly split as follows: 80% for the development process (training) and 20% for the testing process ("out-of-sample" data). In case of SVM there is no need for a validation sample.

For the model development a Gaussian RBF Kernel function previously presented was chosen. This requires selecting the appropriate *gamma*[8] value which controls the threshold that separates the "*affluent*" from "*mass*" customers. A higher *gamma* gives a higher separation between the two classes.

In order to choose the appropriate SVM model in terms of *capacity*[9] (*C*) and *gamma*, several trials were performed. To treat all continuous variables on the same gauge, these were scaled between 0 and 1. The maximum number of iterations was set to 1,000 and the stopping error at 0.001.

Initially, *gamma* was set to 0.05 and the capacity (*C*) was increased until the accuracy on the training or the test sample started to decrease. When $C = 35$, the model reached a maximum detection rate on both samples. Having chosen the capacity (*C*), *gamma* parameter was afterwards stressed in order to see the effect on the accuracy values. The model that brought the highest benefit in terms of detection was *SVM_9*, having *gamma* = 0.17 and *C = 35*. This model led to an overall detection rate of 98.832% on the training set and 97.127% on the test set (all results are presented in Table 4).

Table 4. Trials for SVM selection

Trial	C	gamma	Support Vectors (0)	Support Vectors (1)	Bounded	Train perf. (%)	Test perf. (%)
SVM_1	10	0.05	138	140	256	97.35	96.23
SVM_2	20	0.05	117	124	216	97.889	96.589
SVM_3	30	0.05	109	116	192	98.113	96.768
SVM_4	40	0.05	104	110	177	98.068	96.768
SVM_5	35	0.05	105	113	218	98.158	96.768
SVM_6	35	0.1	93	106	147	98.428	96.948
SVM_7	35	0.15	86	111	123	98.518	97.127
SVM_8	35	0.2	87	104	111	98.832	96.948
SVM_9	35	0.17	86	103	119	98.832	**97.127**

Even if *SVM_8* has less bounded vectors, it is obvious that the detection on the test sample is smaller and this induces the feeling of an over-fitted model with very good performance on the development sample, but with low generalization capacity on out-of-sample data. Therefore, it is always indicated to select the model based on its accuracy on the test sample, and in this case, *SVM_9* satisfies best this necessity.

[8] $gamma = \frac{1}{2\sigma^2}$.

[9] This is the weight given to misclassifications. The higher is *C*, the lower is the generalization capacity and therefore the over-fitting phenomenon appears.

Going further into details regarding the accuracy of the selected model (*SVM_9*), we observe that the accuracy on the "*affluent*" customers is somewhat smaller (83%) than the overall detection rate, but this can still be considered a good detection percentage for this class (Table 5).

Table 5. Confusion Matrix for Test sample using SVM_9

		Predicted	
		mass	*affluent*
Observed	*mass*	**99%**	1%
	affluent	17%	**83%**

6. Conclusions

Both machine learning techniques performed well in the segmentation process. However, even if the overall detection rate on the test sample differs from one approach to the other only by 2.34pps[10], SVM model using *RBF kernel* function clearly outruns the "*affluent*" detection of the MLP using *gradient descent* algorithm. This induces the need of using more advanced algorithms, like *conjugate gradient* or *Broyden–Fletcher–Goldfarb–Shanno* (BFGS), which can solve one of the major problems of *gradient descent*, i.e. the local minimum aspect.

References

Ahmad, A. R.; Viard-Gaudin, C.; Khalid, M.; Yusof, R. 2004. Online handwriting recognition using Support Vector machine, volume A. *IEEE*, 311–314.

Amari, S.; Wu, S. 1999. Improving Support Vector machine classifiers by modifying kernel functions, *Neural Networks* 12: 783–789. http://dx.doi.org/10.1016/S0893-6080(99)00032-5

Amir, F. A. 2001. Bankruptcy prediction for credit risk using neural networks: a survey and new results, *IEEE Transactions on Neural Netwoks* 12(4): 929–935. http://dx.doi.org/10.1109/72.935101

Antkowiak, M. 2006. *Artificial Neural Networks vs. Support Vector machines for skin diseases recognition*. Master thesis in Computer Science, Umeå University, Sweden.

Auria, L.; Moro, R. A. 2008. *Support Vector Machines (SVM) as a technique for solvency analysis.* German Institute for Economic Research, Berlin.

Ayodele, T. O. 2010. Types of machine learning algorithms, in Yagang Zhang (Ed.). *New advances in machine learning.* University of Portsmouth, United Kingdom, 19–46.

Baesens, B.; Setiono, R.; Mues, C.; Vanthienen, J. 2003. Using neural network rule extraction and decision tables for credit-risk evaluation, *Management Science* 49(3): 312–329. http://dx.doi.org/10.1287/mnsc.49.3.312.12739

Boser, B. E.; Guyon, I. M.; Vapnik, V. N. 1992. A training algorithm for optimal margin classifiers, in *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory.* Pittsburgh, Pennsylvania: ACM Press. http://dx.doi.org/10.1145/130385.130401

[10] Percentage points.

Cortes, C.; Vapnik, V. 1995. Support-vector networks, *Machine Learning* 20(3): 273–297. http://dx.doi.org/10.1007/BF00994018

Downs, T.; Gates, K. E.; Masters, A. 2001. Exact simplification of Support Vector solutions. *Journal of Machine Learning Research* 12: 293–297.

Frank, R. E.; Massy, W. F.; Wind, Y. 1972. *Market segmentation*. Englewood Cliffs: Prentice-Hall.

Kiyan, T.; Yildirim, T. 2003. Breast cancer diagnosis using statistical neural networks, in *International XII, Turkish Symposium on Artificial Intelligence and Neural Networks,* University Besiktas, Istanbul, Turkey, 2003.

Knerr, S.; Personnaz, L.; Dreyfus, G. 1992. Handwritten digit recognition by neural networks with single-layer training, *IEEE Transactions on Neural Networks* 3(6): 962–968. http://dx.doi.org/10.1109/72.165597

Kotler, P. 1998. *Marketing management.* Ljubljana: Slovenska knjiga. 832 p.

McCulloch, W.; Pitts, W. 1943. A logical calculus of ideas immanent in neural activity, *Bulletin of Mathematical Biophysics* 5: 115–133. http://dx.doi.org/10.1007/BF02478259

Minsky, M.; Papert, S. 1969. *Perceptrons*. Cambridge: MIT Press.

Noble, W. S. 2006. What is a Support Vector machine?, *Nature Biotechnology* 24(12): 1565–1567. http://dx.doi.org/10.1038/nbt1206-1565

Rosenblatt, F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain, *Psychological Review* 65(6): 386–408. http://dx.doi.org/10.1037/h0042519

Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. 1986. Learning representation by back-propagation errors, *Nature* 323(9): 533–536. http://dx.doi.org/10.1038/323533a0

Ultsch, A.; Korus, D.; Kleine, T. O. 1995. *Integration of Neural Networks and knowledge-based systems in medicine*. Hans Meerwein-Straße / Lahnberge, Marburg.

Vapnik, V. 1995. *The nature of statistical learning theory.* New York: Springer-Verlag. http://dx.doi.org/10.1007/978-1-4757-2440-0

Yoon, Y.; Swales, G. 1993. Predicting stock price performance: a neural network approach, in *Neural Networks in finance and investing*. Chicago, IL: Probus Publishing Company, 329–339.

Zhou, Z. H.; Jiang, Y.; Yang, Y. B.; Chen, S. F. 2001. *Lung cancer cell identification based on artificial neural network ensembles.* Nanjing University. Nanjing, China.

Ion SMEUREANU has graduated the Faculty of Planning and Economic Cybernetics in 1980, as leader. He holds a PhD diploma in "Economic Cybernetics" from 1992 and has a remarkable didactic activity since 1984, when he joined the staff of Bucharest Academy of Economic Studies. Currently, he is a full Professor of Economic Informatics within the Department of Economic Informatics and the dean of the Faculty of Cybernetics, Statistics and Economic Informatics from the Bucharest University of Economic Studies. He is the author of more than 16 books and an impressive number of articles on economic modeling and computer applications. He was also project director or member in many important research projects. He was awarded the Nicolae Georgescu-Roegen diploma, the award for the entire research activity offered by the Romanian Statistics Society, General Romanian Economist Association Excellence Diploma and many others.

Gheorghe RUXANDA is a PhD in Economic Cybernetics, Editor-in-chief of ISI Thompson Reuters Journal "Economic Computation and Economic Cybernetics Studies and Research" and Director of Doctoral School of Economic Cybernetics and Statistics. He is full Professor and PhD Adviser within the Department of Economic Informatics and Cybernetics, The Bucharest Academy of Economic Studies. He graduated from the Faculty of Economic Cybernetics, Statistics and Informatics, Academy of Economic Studies, Bucharest (1975) where he also earned his Doctor's Degree (1994). Had

numerous research visits in Columbia University – School of Business, New York, USA (1999), Southern Methodist University (SMU), Faculty of Computer Science and Engineering, Dallas, Texas, USA (1999), Ecole Normale Superieure, Paris, France (2000), Reading University, England (2002), North Carolina University, Chapel Hill, USA (2002). He is full professor of Multidimensional Data Analysis (Doctoral School), Data Mining and Multidimensional Data Analysis (Master Studies), Modeling and Neural Calculation (Master Studies), Econometrics and Data Analysis (Undergraduate Studies). Fields of Scientific Competence: evaluation, measurement, quantification, analysis and prediction in the economic field; econometrics and statistical-mathematical modeling in the economic–financial field; multidimensional statistics and multidimensional data analysis; pattern recognition, learning machines and neural networks; risk analysis and uncertainty in economics; development of software instruments for economic-mathematical modeling. Scientific research activity: over 35 years of scientific research in both theory and practice of quantitative economy and in coordinating research projects; 50 scientific papers presented at national and international scientific sessions and symposia; 64 scientific research projects with national and international financing; 77 scientific papers published in prestigious national and international journals in the field of economic cybernetics, econometrics, multidimensional data analysis, microeconomics, scientific informatics, out of which eleven papers being published in ISI – Thompson Reuters journals; 18 manuals and university courses in the field of econometrics, multidimensional data analysis, microeconomics, scientific informatics; 31 studies of national public interest developed within the scientific research projects.

Laura Maria BADEA is a PhD candidate in Economic Cybernetics at the Bucharest Academy of Economic Studies, has an MA in Corporate Finance (2010) and graduated the Faculty of Finance, Insurance, Banking and Stock Exchange from Bucharest Academy of Economic Studies (2008). *Fields of scientific interest:* machine learning and other modeling techniques used for classification matters in economic and financial domains, with a focus on artificial neural networks. Scientific research activity: one published article in ISI Thompson Reuters Journal.