DOI: 10.2478/sues-2018-0015

MARKET FORECASTS AND CLIENT BEHAVIORAL DATA: TOWARDS FINDING ADEQUATE MODEL COMPLEXITY

Dan Stelian Deac
Lecturer, Ph.D., Faculty of Economics, Engineering and Informatics, "Vasile Goldiş" Western University of Arad Romania e-mail: dan.deac@uvvg.ro

Klaus Bruno Schebesch
Professor, Ph.D., Faculty of Economics, Engineering and Informatics, "Vasile Goldiş" Western University of Arad Romania e-mail: kbschebesch@uvvg.ro

(Received: July 2018; Accepted: August 2018)

ABSTRACT: Using efficient marketing strategies for understanding and improving the relation between vendors and clients rests upon analyzing and forecasting a wealth of data which appear at different time resolutions and at levels of aggregation. More often than not, market success does not have consistent explanations in terms of a few independent influence factors. Indeed, it may be difficult to explain why certain products or services tend to sell well while others do not. The rather limited success of finding general explanations from which to draw specific conclusions good enough in order to generate forecasting models results in our proposal to use data driven models with no strong prior hypothesis concerning the nature of dependencies between potentially relevant variables. If the relations between the data are not purely random, then a general or flexible enough data driven model will eventually identify them. However, this may come at a high cost concerning computational resources and with the risk of overtraining. It may also preclude any useful on-line or real time applications of such models. In order to remedy this, we propose a modeling cycle which provides information about the adequacy of a model complexity class and which also highlights some nonstandard measures of expected model performance.
Keywords: Aggregate market reaction, individual client behavior, data modeling, deep neural networks, overtraining

JEL Classification: M3, D8, C1, C8.

1. Introduction and motivation

The use of effective marketing strategies in the relationship between seller and buyer depends on a detailed data analysis and the use of this data to forecast collective and individual behavior of economic agents. The data may be collected both at multiple aggregation levels as well as within different functional relations between such agents. In many specific market situations, the mechanisms influencing eventual market success have no consistent theoretical explanation. For instance, it often remains a surprise as to why some products are successfully sold and others are not, despite having seemingly attractive characteristics.

A way out would be to collect very large amounts of detailed behavioral and other market related data, as customers may follow hidden "addictive" behavioral patterns which may be detected, explored and finally exploited – at least to the benefit of vendors. Mapping this data against vending success is an important technique resulting in many types of forecasting and classification models which may also explain purchasing behavior or ranking of alternatives, etc.. However, such big data are predominantely private and are being considered to be an important asset of e.g. technology firms or consultancies. Nevertheless, there is now a clear tendency of increasingly "disclosing" such private data putting them into the public domain. The price to pay are here the obligatory privacy preserving transformations of the data imposed by law to be applied by the data supplier, which may somewhat diminish accuracy and information content.

In the public sphere there are usually small data sets available whose analysis can best be done by means of a classic linear model (or a logistic model for classification which merely transforms a linear relation, Hosmer and Lemeshow (2000)). When, at the other extreme, a considerable amount of data is available, both data analysis and modelling will best be done by means of a neural model (including its many recent variants related to deep learning, view e.g. Bishop (2006), Montavon et al. (2012), Goodfellow et al. (2016)). The downside of the latter approach is (1) large to excessive consumption of computational resources and (2) a considerable uncertainty concerning the adequacy of the resulting neural model complexity. While an optimized linear model (Press et al. (2007)) may indeed be viewed as being the best model for independent additive influence factors explaining a dependent variable (sales, say) there is nothing similar in spirit we can say about a neural model. Aside from the (not so remote) possibility that the model data do not contain all relevant influence variables (McLachlan (2014), Trippa et al. (2015), Gareth et al. (2015)), the resulting linear model can be termed truly optimal in the sense that no better linear model does exist. A similar statement may never be forwarded for a neural model. While it is able to potentially capture any practically relevant type of non-separable and of non-additive (i.e. nonlinear) influences (Bishop (1995, 2006)), one can never be sure if it is also optimal in some sense. Accordingly, it may be (grossly) overtrained, which means it is much more powerful than the noisy data would allow or it is effectively much less poweful than its architecture (number of neurons and parameters) would imply. The latter case can be achieved rather easily by implicitly deactivating whole branches of the network (view e.g. Zimmermann and Weigend (1997), Montavon et al. (2101)).

Hence the task of data modeling in a market environment is to find elements which point towards correcting model complexity: if the data is severly limited use a linear model, otherwise be careful to clarify which is the adequate model complexity for the problem at hand.

In the sequel we will address this problem by finding data models for two conceptually distict data sets which nevertheless stand for two relevant problems in quantitative marketing, namely (1) forecasting a aggregate quantity (vending prices, say) and (2) forecasting the reaction of a new client (buying insurance, say). Some of the available data sets are quite extensive (in the range of 10^4 data points), such that there is substantial a priori uncertainty concerning the adequate model class to use. Data may be more or less redundant, and linear models may suffice. However, there are also different criteria along which to measure modelling success as well as forecasting stability (Bishop (2006)). Hence, there may be situations where a more general modelling approach (e.g. neural models) would deliver the best results.

In order to offer an answer to these questions we proceed by using two empirical data sets with data volumes as announced above, the first being used in order to forecast an aggregate market quantity (regression-based model) and the second one being used in order to forecast a client's choice (classification-based model). We illustrate the gains and downsides for transiting from a linear regression model to a neural network (regression) and transiting from LDA (linear discriminat analysis) to a neural network including a deep learning neural network variant (classification). In doing so we have to explain a few preparatory modelling steps which are required in the case of neural networks but which are not required in linear modelling. In the case of neural networks, we need to (1) preprocess data and continue with the important aspects related to training the networks, i.e. (2) choosing activation function, employing a variant of error backpropagation, and, finally, (3) using cross-validation in order to check for over- or under-training of the resulting models (Trippa et al (2015), Kamisnsky et al. (2017)). We conclude with some comments on the usability of today's technology and data availability, but we also offer an outlook concerning the type of deep neural networks to be used on future still much bigger and more integrated data sets.

2. Research methodology

While in market modeling and in quantitative marketing research there is plenty of scope to use unsupervised statistical learning (e.g. Chollet (2018) for recent examples), i.e. clustering or segmenting persons, products, industries, etc., into groups for deriving recommendations as well as using semi-supervised learing in case of partial availability of label data (see e.g. Bishop (2006), Gareth et al. (2015)

for technical accounts), in the present paper we concentrate on models estimated or trained by supervised learning as is still a more common practice in econometric and marketing applications. Hence, we assume that all input and associated target data are available, there are no missing values in the data, and there are no a priori unknown cathegories (labels) which data subsets are going to be grouped into by subsequent modeling. We propose the following modeling cycle which may be associated with meta-learning about market model building:

1. Find at least two data sets which serve different modeling goals and which by their nature may be complementary, at least in principle
2. Estimate and evaluate basic linear prediction models on these data sets
3. Employ conventional neural networks – subsample the data if required
4. Evaluate alternative, possibly more problem adequate error functions
5. Define and employ an extensive validation procedure on steps 3 and 4
6. Use deep neural networks on full data as a more demanding expert backend

Repeated steps 1 - 6 for different market data and associated predictions tasks may lead over time to a realistic judgement of pertinent statistical modelling possibilities and even to new experimental modeling designs.

With the above stated restrictions in mind we start out from at least two market data repositories with the generic data format being (X, y), where input X is a $N \times m$ - matrix and the target or output y is in general a $N \times p$ - matrix, respectively. A line from X, namely x_i contains the ith observation of input features from the data base. Such an observation might refer to e.g. an object, an action or a person with respective $m > 0$ (supposedly independent) characteristics. Associated with each observation is an output (according to context a target or a response), which may be real- valued (implying to find a regression model on the data) or categorical, like e.g. "yes/no" (implying a classification model).

In general, one would use multiple regression and multi-label classification (hence $p > 1$). In the data examples to follow we refrain from these general cases. We here also refrain from considering time series models where a prediction is conditioned on the (immediate) respective past inputs. Hence, our data model basically poses the question "given an input vector does appear what the most probable output is?"

Linear base models

In view of the general aim of finding well performing prediction models but also of using the least demanding modeling approach (in terms of highly specialized technical expertise and of avoiding excessive use of computational resources) one would naturally start out by using basic linear models for both regression and classification. Given the above definitions a linear (regression) model estimated by minimizing the squared residuals (squared deviations of the predictions from the outputs) reads:

Find a parameter vector w^* which minimizes a user given error function $E(y, f(X, w))$ over all $w \in \mathbb{R}^{m+1}$, with the quadratic error $E(.,.)$

$$E(y, f(X, w)) = \frac{1}{N}\sum_{j=1}^{m}(y_i - f(x_i, w))^{2}$$

and with the model being $f(x_i, w) = w_0 + \sum_{j=1}^{m} w_j x_{ij}$, i.e. it is a hyperplane in the m-dimensional feature space. A linear binary classification which assumes $y_i \in \{0,1\}$, say, is a hyperplane perpendicular to the connecting line of barycenters (centers of gravity) of the data points of the two classes. If $b_0 = \sum_{i=1}^{N(y=0)} x_i$ is the barycenter for observations with $y_i = 0$ and $b_1 = \sum_{i=1}^{N(y=1)} x_i$ for the remaining observations with $y_i = 1$, an optimal separating hyperplane will sit somewhere on the line $\lambda b_0 + (1-\lambda)b_1$, with $0 < \lambda^* < 1$ and with the mentioned perpendicularity condition of $\sum_{j=1}^{m}(b_0 - b_1)w_j^* = 0$.

Such linear optimality schemes are computed cheaply. But they strictly do apply just in case of data generated under Gaussian distributions with well separated means (Press et al. 2007). The ideal case is to find a linear model such that the remaining errors (unexplained residuals) are themselves normally distributed, meaning that there is no more systematic dependency to extract from the data. As soon as these assumptions concerning the data and the models are not verified the question of how to find an adequate (performant, parsimonious) data model is fully open again.

Conventional (shallow) neural networks

The aforementioned conditions may be rarely satisfied in practice. A valid but pragmatic argument in favor of linear models in econometrics and quantitative marketing is that of data scarceness. While still true for many interesting sub-domains this is changing fast. A subtler argument refers to the judgement that economic data tends to be heavily corrupted by different noise sources and as such is not very reliable anyway. Repeated waves of at best moderately successful attempts to model (high frequency, etc.) financial data streams (e.g. Zimmermann and Weigend (1997)) which come with the promise of being extremely lucrative for the modeler underline this conclusion.

However, the general data landscape for market related modelling is much more diverse than financial times series or trading and the prediction tasks are equally very different in nature. Additionally, some data like recorded opinions, actions or decisions, etc., are not easily associated with "noise" in any classical sense. In all these cases and when we suspect that a linear model is suboptimal for some reason we may switch to a much more flexible modeling tool like neural networks. Neural networks can be used for regression and classification alike (and for many more different tasks as well). They are highly parameterized nonlinear statistical models and therefore capable to model very different behaviors by potentially producing a very large spectrum of different input-output mappings (for a recent account see Montavon et al. (2012)).

A conventional neural network has these capabilities but it requires in general a huge (computational) training effort which is further potentiated by the necessity of cross-validation (Bishop (1995, 2006)). The latter is a procedure for actively avoiding overtraining (i.e. showing good performance in training but being a poor forecaster on unseen or new examples). Also, training may be practically ineffective for bigger data sets. The upside is that they do not need extensive technical knowledge and many software packages (e.g. R, SPSS, Eviews, etc.) do provide user-friendly basic tools. Their basic mode of operation goes as follows: starting out from a user chosen error function as in the case of the linear model, $E(y, f(X, w))$ which we intend to minimize, the main difference concerns the model *f* (.,.). We will refer to the standard feedforward neural network for supervised learning. This model is a hierarchical composition of simple nomlinear nodes (the "neurons") which are interconnected like nodes of a directed weighted graph without cycles. The inputs are the sinks of such a graph and the output is the goal. The graph (or neural network) incidence matrix is such that only a (lower, say) triangle contains connections. Shunting connections are also not used, such that the networks look like sandwitched neurons. Figure 1 is depicting the incidences of a 5:3:2:1 neural network. There are 5 inputs followed by 3 neurons in a first layer (the 3x5 shaded area) which is followed by a second layer of neurons (the 2x3 shaded area) and finally by an output neuron taking the outputs of the neurons from the second layers as their inputs. Every neuron takes as many weighted inputs as there are outputs in the previous layer.

	Inputs	**Neurons**
Inputs x		
Neurons	**1 1 1 1 1** **1 1 1 1 1** **1 1 1 1 1**	 **1 1 1** **1 1 1**
Output *f*(.)		**1 1**

Figure 1: A small feedforward neural network incidence matrix

The generic formula for e.g. neuron number 4 (first neuron in the second layer) is given by

$$\sigma_4(.) = \sigma\,(\,w_{40} + w_{41}\,\sigma_1(.) + \,w_{42}\sigma_2(.) + w_{43}\sigma_3(.)\,).$$

In this way one recursively reaches the input data (via $\sigma_1, \sigma_2, \sigma_3$). This network has a total of 3x5 + 3 plus 2x3 + 2 and 2+1 (=29) parameters $w_{ij} \in \mathbb{R}$. As is quite obvious the number of of these parameters grows fast in the number of inputs and neurons. From the viewpoint of standard statistical modelling (e.g. Hosmer and Lemesow (2000)) this may seem highly questionable, especially for a given constant number of observations. From the point of view of machine learning (Bishop (2006)) only consistent prediction performance counts – much less the means by which this may be achieved. An argument in favor of this is the very nature of a typical neuron transfer function $\sigma(.)$. Such a function belongs to a class of "squashers", a much-used representative of which is $\sigma(c+u) := \tanh(c+u)$, with c a real constant and u a real variable. This function converges to -1 for large negative values of u, grows monotonuously, and converges to +1 for large positive u. The constant c shifts the zero location to the left or to the right. It thereby allows for itself a series of different responses. More to the point, for increasing values of $|u|$, $|u|>|c|$, the function $\sigma(u)$ rapidely becomes insensitive (i.e. $\frac{d\sigma(u)}{du} \to 0$). As most types of parameter adaptation ("learning") in order to minimize the error function over a training data set draw on exactly this sensitivity of the neurons to incoming signals (e.g. if an incoming signal contributed to an error at the output its strength and direction may be weakened or reversed and vice versa) a neuron can herewith be steered out of its sensitive domain. Hence a formally very complicated looking network may effectively implement a much simpler functionality. With virtually no resource limits on computation the right configuration can eventually be reached. Hence, the downside of this property of neurons and conventional neural networks is twofold (1) increasingly insensitive neurons slow down learning very much, and (2) a time-consuming cross-validation procedure may be required in order to reach acceptable results.

The main purpose of cross-validation in general is to test the out-of-sample prediction performance, i.e. predicting new data that was not seen during learning a model of any kind in order to identify issues related to overtraining (Kaminski et al (2017)). The cross-validation error is essentially the mean error resulting from a large number of models (re-)trained on data where "k out of N" data points were randomly removed. The error contribution of each such model, however, is measured on exactly those k data points. Even for k=1 this error provides a consistent estimate of the error to be expected on truly independent data sets (i.e. on unknown future data occurring within a real-life application).

Deep neural networks

A deep neural network (DNN) has a potentially large number of layers (view figure 1) and, consequently, a very large number of neurons. DNN used for image classification use millions of examples, thousands of input features (e.g. pixels) and tens of millions of neurons and parameters, e.g. consult Goodfellow et al (2016) and Liou et al. (2014). They hope to isolate patterns or object features from images by mapping a large number of partially overlapping picture segments to the consecutive layer of neurons in a chain which may contain hundreds of layers. The theory concering DNN is not new (Hinton et al (2006) and the references therein). However, by employing top level information technology with very high computational throughput, such applications enjoy remarkable success since approximately 2012 onwards. In our market modelling applications, we can't make use of information from geometrical neighborhoods (at least for now, i.e. without fusing conventional data with multi-media content); we use only certain aspects of DNN and their more demanding technology for cross-checking the quality of our models. In principal we use the following features of DNN: (1) the ability to cope with much bigger data sets owing to advanced implementations by computer code Keras (view e.g. Chollet (2018)) using backends like Theano (a code for deep learning by Montreal University, view e.g. Bourez (2017)) and/or Tensorflow (a code forwarded by Google, view e.g. Bevan (2018)). Futhermore, (2) the ability of auto-regularizing inititially over-specified neural networks leading to problem-adapted model complexity which tend to avoid over-training. This was traditionally achived by adding complexity penalties to the error function $E(y, f(X,w))$, mostly by restricting the "mass" of active parameters and /or by restricting the functional complexity (e.g. the curvature) of the input-output map $f(X,w)$. The updated goal of model building would then be to find a parameter vector w^* which minimizes the weigted sum of the error, the parameter mass and that of the the input-output variation, namely

$$E\big(y, f(X,w)\big) \; + \; a\sum_i |w_i| \; + \; b\,G\Big(\frac{\partial f(.,.)}{\partial x}, \frac{\partial^2 f(.,.)}{\partial x^2}, \ldots \Big) + \ldots$$

with hyperparametes $a, b \geq a$, to be set by the user (Hirasawa et al. (2000)), weighting, respectively, the parameter mass, the penalization of this reducing the specification risk (Bishop (1995)), and $G(.)$, a positive function increasing in its arguments, penalizing excessive variation and curvature (view e.g. Schebesch (2003)). This has the aspect of a well-defined optimization problem and would also have the advantage of not throwing away observations for the purpose of validation, but it is nevertheless hard or at least tedious to realize in computational practice.

A more practical way of neural modeling with regularization is Dropout. In principle, this idea has been in use for a long time in various neural network approaches and is known as "pruning and re-activation cycle" (Bishop (1995), Zimmermann and Weigend (1997)). However, in the case of DNN the drop-out method works especially well: A drop-out probability can be chosen for each layer. Drop-out randomly removes nodes from the network along with all their input and output connections and thus prevents over-training. A sparsely connected neural network emerges that is well adapted to the prediction task

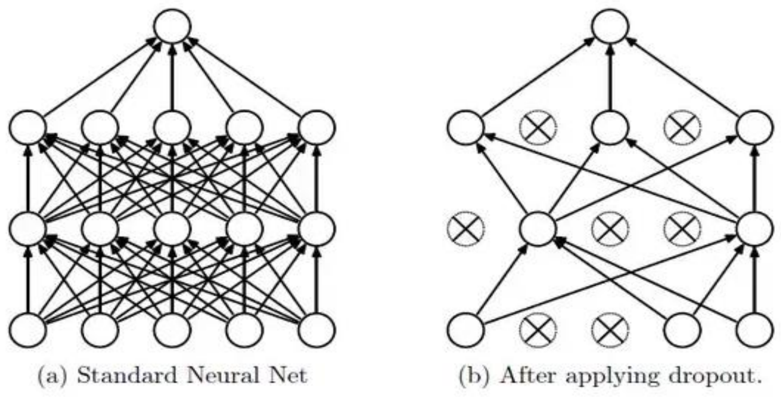


Figure 2: Basic principle of drop-out: deactivating some neurons

The process may be repeated over the training epochs following a mechanism which may be complicated in detail but serves the overall purpose of thinning out the network without following an expensive gradient-based recipe. The remaining (active) neurons and their incoming parameters are updated by automatic differentiation (Baydin et al. (2018)), an efficient version of using symbolic derivatives. By and large the modern implementations of DNN draw on ideas expressed by the above regularization goal, especially as the provide new and very efficient implementations of tensor-valued second derivatives.

Modeling aggregate market behavior by regression-based data models

For the comparative analysis of a regression-based task to be done between a classical linear model and a classical neural network, we chose a set of data containing N=506 observations and a total number of 14 variables (features) in total. This data set contains information gathered by the US Census from several locations within the Boston area in Massachusets and was later included in the StatLib archive (Repo1 (2018)) and is now publicly available through the R-CRAN data archieves as well (Rcran (2018)).

The motivation for the selection of this set of variables for the task of house price prediction as a function of social, environmental and economic factors is documented in Harrison and Rubinfeld (1978). The 14 features (attributes) in each observation recorded at a location of the region are:

1. CRIM - per capita crime rate by city
2. ZN - the proportion of residential land for lots of over 8,000 square meters.
3. INDUS - the percentage of land occupied by non-retail business on the city.
4. CHAS - Variable dummy Charles River (1 if the river passes through the city, otherwise 0)
5. NOX - the concentration of nitrogen oxides (parts per 10 million)
6. RM - the average number of rooms per dwelling
7. AGE - the proportion of owner-occupied dwellings built before 1940
8. DIS - the weighted distance of five Boston employment centers
9. RAD - accessibility index for radial motorways
10. TAX - the total property tax rate of $ 10,000
11. PTRATIO - the student-teacher report by city
12. B - 1000 (A - 0.63)^2, where A is the proportion of Afro-Americans in the locality
13. LSTAT - percentage of the population with lower social status
14. MEDV - Median value of owner-occupied homes (in steps of $1000)

(source: http://lib.stat.cmu.edu/datasets/boston, slightly adapted)

The last variable MEDV was chosen as a prediction target and the rest are treated as independent input variables per observation. Hence, we have a number of inputs *m*=13 and a sibgle real valued output, *p*=1, completing thus the basic description of our modelling data (X, y).

At first, we are looking for a linear model, formally expressed by $y_i = f(x_i, w,) + \epsilon_i$, with $i = 1, \ldots, N$, which minimizes the mean squared error $1/N \sum_i \epsilon_i^2$. In the case of a linear model there are exactly 14 parameters of the linear regression with the components: $w_0, w_1, \ldots, w_{13}$, with w_0 the intercept, which is dictated by the fact that we have 13 separable inputs. In the general case when fa neural network is for example, the vector w may have arbitrary length. Figure 3 depicts the linear correlations between the inputs as a histogram. Excluding the last bar (diagonal of the correlation matrix) and may conclude by visual inspection that there is no clear pattern leading one expect an optimal outcome from a linear model which separates the contribution of each input. The seems to be an execption which we will wait to identify by the linear model. Already at this point one may expect that non-linear structures (such as those of neural networks) may capture more of the nonseparable influences in the data.

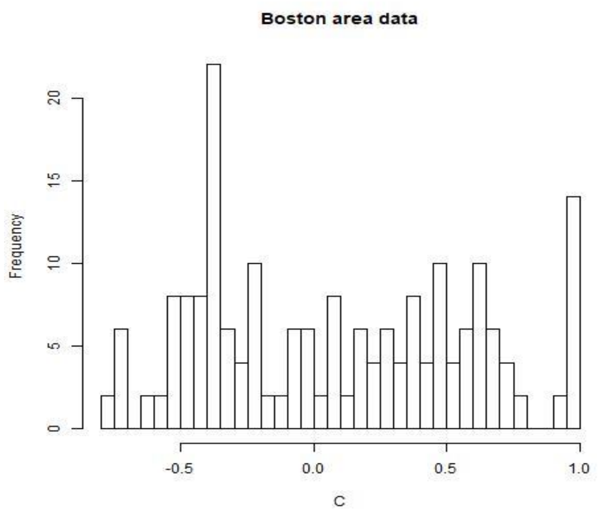


Fig.3 Distribution of linear correlations between the 14 variables.

To facilitate comparison, we begin with the estimation of a linear model (Gareth et al. (2015)), which can be trained in a matter of seconds by a modern statistical modeling software tool (e.g. R). This will be followed by the training of non-linear models. A major downside of the latter is the necessity of a cross-validation procedure in order to avoid over-training. Also their training-validation cycle duration can increase very sharply with an increasing number of observations.
The estimation of the linear model by the appropriate module from the software tool R (i.e generalized linear modelling – glm()) results in the following output (Figure 4):

```
glm ( formula = medv ~., data = data)
Deviance Residuals:
     Min           1Q         Median        3Q          Max
-15.5944739  -2.7297159    0.5180489    1.7770506  26.1992710

Coefficients:
   Estimate    Std. Error    t value
( Intercept) 36.4594883851 5.1034588106  7.14407
crim        -0.1080113578       0.0328649942 - 3.28652
zn           0.0464204584 0.0137274615  3.38158
indus         0.0205586264 0.0614956890        0.33431
chas         2.6867338193 0.8615797562  3.11838
```

```
nox         -17.7666112283 3.8197437074 - 4.65126
rm            3.8098652068 0.4179252538   9.11614
age           0.0006922246 0.0132097820   0.05240
dis          -1.4755668456 0.1994547347 - 7.39800
rad           0.3060494790 0.0663464403   4.61290
tax          -0.0123345939 0.0037605364 - 3.28001
ptratio      -0.9527472317       0.1308267559 - 7.28251
black         0.0093116833 0.0026859649         3.46679
lstat        -0.5247583779 0.0507152782 -10.34715
Pr (> | t |)
( Intercept)  0.00000000000328344 ***
crim         0.00108681 **
zn            0.00077811 ***
indus        0.73828807
chas          0.00192503 **
nox           0.00000424564380765 ***
rm           <0.000000000000000222 ***
age           0.95822931
dis           0.00000000000060135 ***
rad           0.00000507052902269 ***
tax           0.00111164 **
ptratio       0.00000000000130884 ***
black         0.00057286 ***
lstat       <0.000000000000000222 ***
---
Signify. codes:    0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '' 1
```

Fig. 4 Linear model statistics for the housing data prediction

Indeed, there is a single variable, namely `nox` (the concentration of nitrogen oxides) with a high negative influence on the model output. Together with the very high estimated value of the intercept in comparison to the rest of the estimated model parameters the results from Fig. 4 underline the impression that the linear model may be suboptimal in being near degenerate with regrad to all the 13 input dimensions.

Training a conventional neural network as a next step will help clarify if this judgement about the hitherto estimated linear can be justified. The neural configuration we subject to the training procedure 13:5:2:1, i.e. the 13 given inputs, five neurons in the first layer, two neurons in the second layer, and one output. The motivation for this (heuristically) chosen configurartion is simply to force the model to compress information in two steps, reducing the number of neurons each time by at least one half. This model has already (13x5+5) + (5x2+2) + (2+1) = 85

free parameters, which taken together with the tanh(.)-neurons used in the hidden layers results in a nominally much more powerful data model then the linear benchmark.
For all neural network modelling, it is highly recommended to standardize variables as minimum data preprocessing (view Kotsiantis et al. (2006)), for instance by mapping the actual range of each variable uniformly into the interval [0,1] (such transformations should also be reversible for later analysis). After standardizing the data, we employ a more recent, user friendly R-package for conventional neural networks. The neural network training will be performed with error-backpropagation by minimizing the mean squared error (by default).

```
nn <- neuralnet (f, data = xtrain_, hidden = c(5,2),
linear.output = T)
```

Fig. 5 Training a conventional neural network with an R-package resultin in model `nn`.

Using this model, we calculate the perdictions on the (held out) test set examples and the mean squared error of the model on the test set. We compare the two average square errors on the test set produced by the linear model (`lm`) and the neural network model (`nn`):

```
print (paste (MSE.lm, MSE.nn))
[1 ] " 21.6297593507225     10.1542277747038 "
```

Fig. 6 Comparison of linear (left) Neural network model errors (right)

Upon comparing the two errors the superiority of the predictions by the neural network clearly results. Another way of viewing the extend of the diffrerence between the forecasts of the two models on the test is by comparing the scatter plots of actual against the predicted outputs for each model. The closer the points group around the diagonal the better are the prdictions (perfect predictions imply all the points are exactly on this diagonal). The result of this comparison is shown in Fig. 7, and it indicates that the locations of the points produced by the neural model are grouped more closely around the diagonal (lhs inset of Fig. 7) than those produced by the linear model (rhs inset of Fig. 7). In this example, the difference is clearly noticed, by visual inspection.

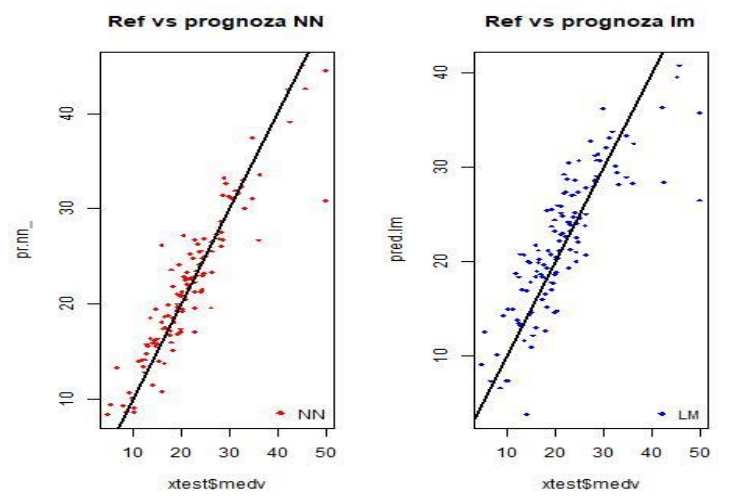


Fig. 7: Comparison of the scatter plots "predicted against true values" for both models

In general, in the case of neural networks, a single training cycle (validated on a single test set) does not provide sufficient confidence in the expected prediction performance of the model. In order to obtain a more valid estimate we proceed by using a process of cross-validation. This process is in essence a loop over *k* >> 1 "newly parametreized" neural networks obtained by training them, respectively, on a data set which left out one or more observations chosen at random.

```
set.seed (450)
cv.error <- NULL
k <- 100
for ( i in 1: k) {
index <- sample ( 1: nrow (data), round (0.9 * nrow (data)
train.cv <- scaled [index,]
test.cv <- scaled [-index,]
nn <- neuralnet ( f, data = train.cv , hidden = c (5,2),
linear.output = T)
  prnn <- compute (nn, test.cv [, 1: 13])
  pr.nn <- pr.nn $ net.result * (max (time $ MEDVE) -min (time $
MEDVE)) + min (time $ MEDVE)
  test.cv.r <- (test.cv $ medv) * (max $ data) $ min (data $ medv)
-min (data $ medv)
cv.error [i] <-sum ( (test.cv.r - pr.nn) ^ 2) / nrow (test.cv)
}
```

```
mean (cv.error)
cv.error
```

Fig. 8 Validation procedure for neural network models

The expected prediction performance is then given by the average error measured over all the left-out observations (they serve as a sort of collective test set). We implemented the following validation procedure in R which leaves out 10% of the observations chosen at random. This loop produces 100 newly parameterized models saving the respective errors in the vector named `cv.error` (view Fig.8).
For the linear model, a similar validation procedure returns an expected error of 23.71698765, which is slightly worse than the error produced over the single model at the beginng of this section. The validation procedure depicted in Fig.8 produced over the 100 neural networks an expected error of 12.50847839. The difference between the two values confirms the superiority of the neural model in a statistically most robust way.
The slightly worse performance of both model types under the validation procedure is to be expected since in any case, the training sets of the models are by 10% shorter than those from the former experiments. An interesting additional in-formation for the appreciation of modeling risk by neural networks on this specific data can be seen in the error distribution on the different neural models produced during the validation procedure:

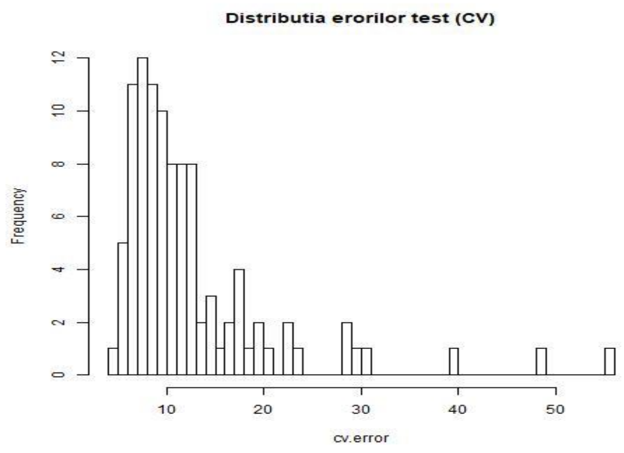


Fig. 9 Histogram of error distribution over 100 neural models

The visualized information confirms the modeling quality of the neural networks, indicating concentrated of errors around a low value. At the same time of highlights the chances of being deceived by a single or a too low number of modelling attempts.

Modelling individual client behavior by classification-based data models

We now turn to the case of a much bigger data set which requires a classification model in order to predict the buying decision of clients. Such classification models are often used in quantitative marketing and in marketing research. They have a wide area of applicability in predicting differrent kinds of actions (including fraud, churn, etc.), attitudes and opinions. As versions of supervised learning they are instrument in helping to build recommender systems and other personalized servi-ces. The data we use in our experiments to be described in the sequel are available from the UC Irvine Machine Learning Repository (UC stands for University of California). They contain selected information about a direct marketing campaign of a Portuguese banking institution. It was used by Moro et al. (2014) in the context of telemarketing decision support. The goal of this campaign is to persuade customers to underwrite a term deposit. This dataset was technically obtained by downloading the file `bank-additional-full.csv` contained in an archieved file `bank-additional.zip` from the named repository (Repo2 (2018)). The raw data contain N=41,188 observations (past clients) and each client is described by m=20 features including a feature (decision recorded in the past) which will be used as the dependent variable. Table 1 contains the description of these variables.

The last entry from Tab. 1 is the categorical variable ("yes / no") that expresses the purchase option and will be the prediction target. Using R, we convert a selection of categorical data (which were represented by alphanumeric strings) into numerical equivalent representations. Similar procedures transform the dependent variable `y_train,` as well as those for the test set, namely `x_test` and `y_test.` We chose the traing set to contain 35009 observations (clients) and the complementary test set to contains 6179 observations (15% of the total of 41188).

Table 1

Description of client variables from the financial marketing classification task

Variable name	Description	Type
`age`	Age of the client	Numerical
`job`	Occupation of the client	12 categories
`marital`	Marital status	4 categories
`education`	The client's level of training	9 categories
`default`	Indicates whether the customer has a short-term credit	3 categories
`housing`	Indicates whether the client has a home loan	3 categories
`John`	Indicates if the client has a personal loan	3 categories

`contact`	Type of contact communication	2 categories
`month`	The month in which the last contact was made	12 categories
`day_of_week`	The day the last contact was made	5 categories
`duration`	The last contact time in seconds	Numerical
`campaign`	The number of contacts made during this campaign for this client (including the last contact)	Numerical
`pdays`	The number of days the customer was last contacted in a previous campaign	Numerical
`previous`	The number of contacts made before this campaign for this client	Numerical
`poutcome`	The result of your previous marketing campaign	3 categories
`Empvarrate`	Rate of change in occupancy rate (quarterly indicator)	Numerical
`conspriceidx`	Consumer price index (monthly indicator)	Numerical
`Consconfidx`	Consumer confidence index (monthly indicator)	Numerical
`euribor3m`	Euribor 3-month rate (daily indicator)	Numerical
`Nremployed`	Number of employees (quarterly indicator)	Numerical
`y (target)`	Indicates whether the customer has completed a term deposit	Binary (`yes/no`)

Source: https://archive.ics.uci.edu/ml/datasets/Bank+Marketing and own processing

Our modeling experiment starts with estimating a linear discriminant model (LDA) for the binary variable *y*. For the estimation of a linear discriminant classification model (McLachlan (2014)), the statistical software R provides the `lda()` function that produces a categorial output (a non-numeric object). This function also requires a variable **`prior = (0.7,0.3)`** indicating the relative weight of the binary classes (number of clients with decision y="no" against the number of clients with decision y="yes"). For example, in a binary classification problem where we distinguish two labels, i.e. naming events or classes, these may be unevenly represented or the cost a misclassification may differ widely between the classes. A *prior* for the LDA has the effect of translating the separating hyperplate between the two classes across the connecting line between the two class centers (barycenters).
In our data we have 26257 observations for training (the remainder is reserved for the test) and there are 23250 observations with y = 0 (named frequent events hereafter) and 3007 observations with y = 1 (rare events) resulting in a prior of (0.88, 0.12). After determining the optimal separating hyperplane, we test the model prediction on the test set with 8752 test observations.
Figure 10 shows the predictions on this test set sorted by the size and the type of their errors.

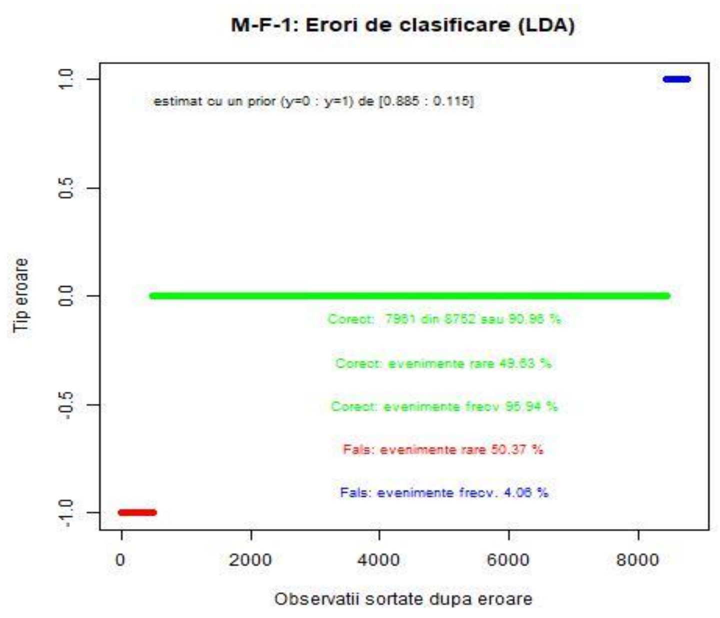


Fig. 10 Two types of classification errors for the LDA model on the test set

For an LDA on our data classification problem, the forecast can only be 0 or 1. The resulting error may occur for a frequent or rare case. By convention, we assign the error that is produced for a frequent case by +1, and those produced for a rare case by -1. The long middle line in Fig.10 at zero are the cases for which the model forecast is correct. The perdiction quality (hit-rate) surpasses 90% overall, but there is a very large error rate in the case on rare events. If the incorrect prediction of a rare event is very expensive, this will obviously overshadow the overall good-looking result.

A neural network for binary data classification

Following a similar goal as in the previous section, namely to move from a linear model to a more flexible nonlinear one, in our first attempt of running a neural network on the same set of 26257 training cases (observations), we confronted a substantial computational problem. After choosing an objective function appropriate to binary classification models, namely cross-entropy, we notice that a relatively simple neural network with the connection structure 20:5:1 already consumes an unacceptable calculation time (over 15 hours terminated by manual interruption).

Consequently, we decided to use a sample of 500 randomly selected observations that reproduced approximately the relative frequencies between frequent and rare events. For this very small training set, a neural network with performance comparable to the LDA was obtained. However, it also contains more features, owing to the continuous outputs in the range [0,1] generated by this type of model as proxies for the categorical discrete outputs {0,1}.

Figure 11 contains two insets which illustrate the representativeness of the reduced training set for the two input variable `age` and `job` (see also Tab. 1). The curve with sorted values shows the whole training set while the dots depict the 500 individuals chosen at random for the reduced (effectively used) training set. The selected points cover the entire value distributions to a reasonable extend.

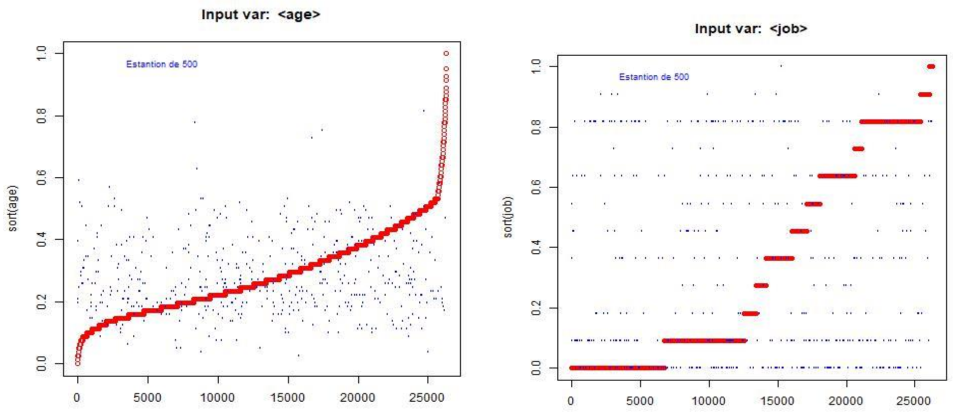


Fig. 11 The age and the job variables from the input data

The classification performance of the resulting neural network is finally measured on the test set comprising the 8752 test observations. In this way, the out-of-sample prediction results of the neural network can be compared directly with that of the LDA model. First, we make an image with an *empirical* progression as it would be in a real application.

Figure 12 illustrates that most of the forecasts for frequent events are correct and the forecast of rare events is of a very low quality. An output less than 0.5 results in a "0" and otherwise "1" prognosis. For instance, it is noted that the forecast for the behavior of client no. 30 is incorrect.

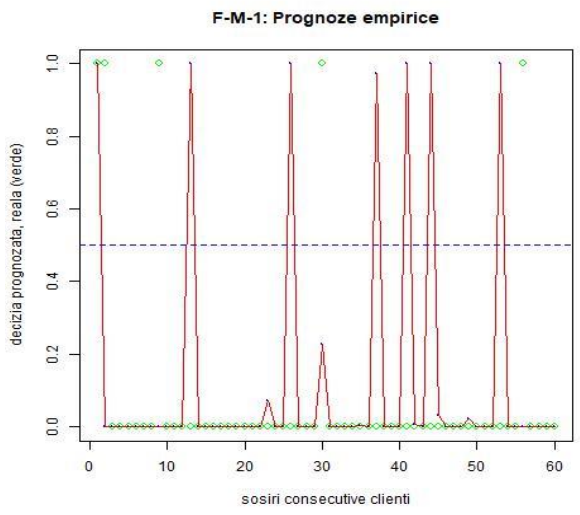


Fig. 12 A randomly selected series of 60 forecasts (red lines) confronted with the true value of the target variable (green bulbs)

The following figure (Fig.13) further illustrates the forecasting quality of the neural network described above on the different sets of errors. In analogy with the LDA case, the neural network output on the 500-observations training set, produces accurate classification and distribution with a focus on frequent events. The difference lies in the fact that the neural network produces a continuous output that can calculate different prediction performances for a variable cut-off, as will be explained next. In figure 13 we depict in the lhs image the prediction performance on the whole training set with cut-off (which varies in this case between -0.5 and +0.5) followed in the same vain by the prediction performance on the set of rare events depending on cut-off (middle image) and finally, the prdiction performance on the set of frequent events (rhs image). From these images, one observes that the prediction performance on one set may increase to the detriment of performance on the opposite set. The cut-off value to be chosen for a final prediction model depends on the relative cost of a misclassification of a set with a given label (or class).
These possibilities may be expressed handily by calculating the ROC (Reciever Operating Characteristic) curve that indicates that the utility of the cut-off variation is quite limited for an LDA model. As seen in figure 14 (lhs inset) the purely discrete nature of the output of the LDA model precludes the use of a cut-off variation as neural classification models (rhs inset).

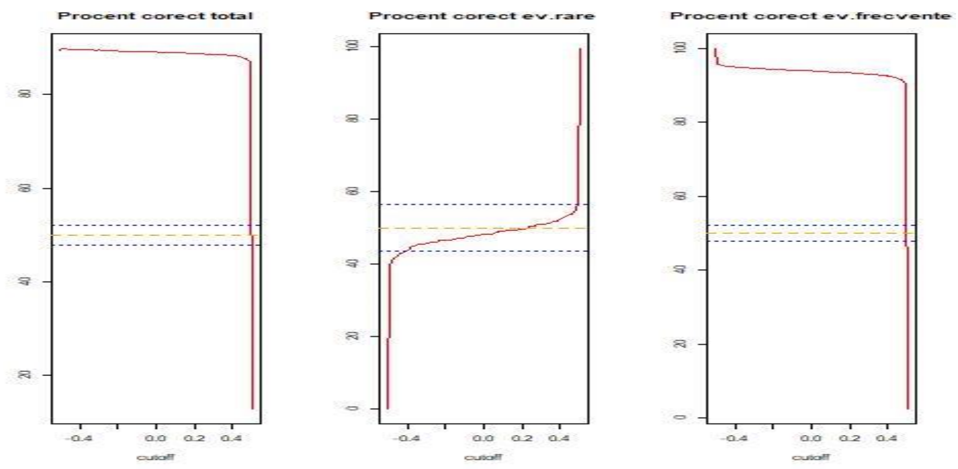
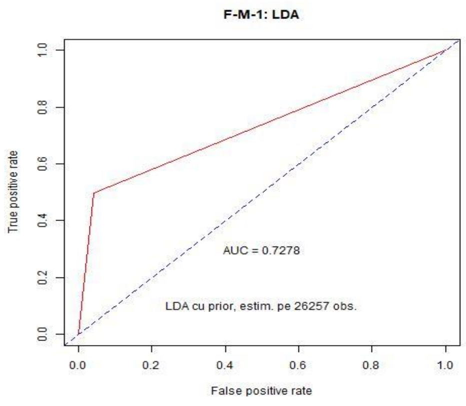


Fig. 13 Prediction performances on different sets with 95% confidence intervall



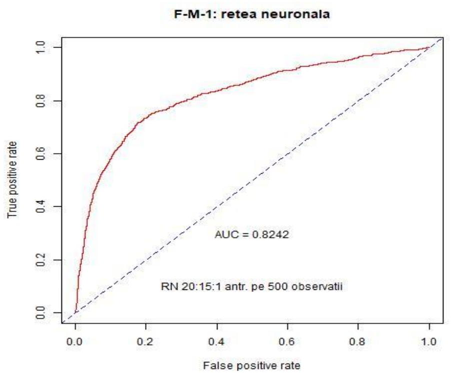


Fig. 14 The ROC curve for the LDA model (lhs) and for the neural network (rhs).

The area under the ROC curve (referred to as the AUC in the literature) indicates prognostic performance over the variation of the cut-off parameter. Consequently, according to the AUC criterion, the area generate by the neuronal network is bigger, hence the neural model may be considered superior to the LDA model. The diagonal is the behavior of what purely random classifications would generate, which also has the lowest possible AUC.

Although, according to the basic criteria for measuring the predictive performance of the classification models (hit rate, etc.), the two models do not appear to be significantly different, considering a more complex but empirically relevant criterion like AUC leads to a clear conclusion that more flexible models like that of neuronal type are worthwile considering.

Towards deep neural networks with marketing applications

As argued before in marketing application we still do not have the kind of data (numerical plus multimedia) which could eventually make use of the full potential of DNN (view e.g. Goodfellow et al (2016), Ioffe et al. (2015)). In our present context we propose to use certain aspects of DNN in order to cross-chek the prediction performace obtained by the classification models on the bigger data set. Our conventional neural model could not be trained on the full data set. DNNs however, are by no means limited by this data size. First, we set up an equivalent 20-input neural network with certain features of a DNN. We use two hidden layers with the `tanh()` activation functions and with 64 neurons in each layer and a two dimensional "softmax" output with values between [0,1] as proxies for the two dummy-encoded outputs (0,1) for *y*="yes" and (1,0) for *y*="no" and minimize a cross-entropy objective function.
The code snippet from Fig. 15 shows our setup of a high performance DNN using the modeling language *Keras* called from within a *Phyton* environment. Note that this DNN has at least 5632 free parameters and is probably oversized.

```
data_dim = x_train.shape [1]
nb_classes = 2
np.random .seed (1047)
model = Sequential ( )
model.add ( Dense (64, input_dim = data_dim, kernel_initializer =
'uniform'))
model.add ( Activation ('tanh'))
model.add ( Dropout (0.5))
model.add ( Dense (64, kernel_initializer = 'uniform'))
model.add ( Activation ('tanh'))
model.add ( Dropout (0.5))
model.add ( Dense (nb_classes, kernel_initializer = 'uniform'))
model.add ( Activation ('softmax'))
model.compile (loss = 'categorical_crossentropy',
Optimizer = 'SGD'
metrics = [ "accuracy"])
```

Fig. 15 DNN setup with Keras with 64 neurons per hidden layer and a Dropout of 0.5.

Parameter $0 \leq \text{Dropout} \leq 1$ is essential for training a potentially oversized DNN. In theory, it provides for a suitably truncated and optimized data model (Srivastava et

al. (2014)), and it helps to avoid overfitting (a highly deceptive effect with devastating consequences on real applications if it is not remedied).
We trained this DNN model on the full dataset on multiple platforms (Linux, Windows) and by using multiple combination of high performance tools, i.e. Keras with Theano backend, and Keras with Tensorflow backend, respectively. The results obtained are very similar to those obtained by the neural network trained on the 500-observations subsample as described in the prvious section. For the time being this result is encouraging, in that a grossly overspecified DNN successfully auto-regularizes to find a data model well adapted to a given prediction task.

3. Conclusions and outlook

For our data modeling experiments, we used two sets of empirical data: one of urban socio-economic context and the other from a financial banking application. As modeling cathegory proxies, they stand for aggregate market forecasts and for individual client behavior forecasts, respectively. From a technical viewpoint they cover both regression and classification. They represent a broad spectrum of modeling approaches from classical linear models to one of the latest machine learning tools, namely deep neural networks.
We start the analysis with a classical linear regression model for the first data set and then gradually go on to more demanding modeling. In this case there is a suspicion that the linear model is sub-trained, therefore it cannot express many of the dependencies existing in these data.
We then switch to the first nonlinear model, namely a 13:5:2:1 neural network that leads to significantly better results on the first dataset, both on the training set and even on the reserved test examples. This confirms that neural networks have a major role to play in those economic applications, for which we have a sufficiently large number of observations but and also a large number of features (or inputs).
The second dataset is referring to a client behavior forecasting problem from financial services marketing uses quite a large number of observations (41.188 cases) and has 20 input variables. The modeling goal is to predict the buying decisions of clients and it is hence cast as a binary classification problem. In the case of financial marketing data, we found that the original training set is too large for a classic neural network and we therefore use a random sample of only 500 training observations, while the test set remains the same as for the previously used linear classification model. The trained neural network has a simple 20:5:1 architecture (a single layer hidden with 5 nonlinear neurons). This simple classification model uses the cross-entropy objective function which in this case can be trained within seconds.

In contradistinction to the modeling of the first data set we now we find that the linear and the neural model do not produce major differences with regard to forecasting performance measured as basic overall hit rates (i.e. listing the correct forecasts for all observations from the test set). However, continuing with a more specific analysis focused on classification models that highlights the relationship between different types of forecast errors produced by these models reveals some significant differences. Treating errors that are related to the class or category to be predicted (false negatives, false positives) differently can be very important in practice since they may relate to vastly different misclassification costs: it may be more expensive to incorrectly predict a rare decision.

From this analysis it follows that by varying the cutoff value, the performance of the neural network predictions can be maded to favor a particular type of event (decision) than would be the case in the classification linear model. This much greater flexibility of a neural classification network is highlighted by the ROC curve and the AUC that indicates a clear superiority of the neural network.

In order to finally demonstrate the usefulness of the most recent addition to data modeling namely that of deep neural networks, we did a series of extensive experiments on the second data set using efficient new implementations for these advanced modeling methods.

We start by training a batch of "big enough" deep neural networks on the entire training data set. Hence, we expect them to be over specified (in terms of number of parameters, neurons, layers, etc.). In case of standard neural networks this would lead to prohibitively long training durations and to grossly over trained models (usually without any beneficial effects on out-of-sample prediction performance). In our case of deep neural networks, this effect cannot be observed. The basic prediction performance of every deep neural network is very similar to that of the linear and the conventional neural model. This feature implies the comfort of starting from an over-specified model, and then automatically finding a properly functionally truncated neuronal neural model with acceptable training duration thus automatically avoiding overtraining. It is also confirming that, at least in terms of basic classification performance, it is unlikely to find a significantly better model than the linear one. Such a conclusion may or may not be valid for other data. Hence a modeling cycle like ours is very important for many electronic marketing applications where public and private data become accessible in large quantities. However, broad data availability is also associated with (1) no well-motivated hypotheses for possible dependencies in the data and (2) uncertainty about whether there really is any "value" in the huge data collections or if - instead - a much smaller data sample would suffice.

We hence expect methods and procedures that are inspired by large data sets, and which make use of machine learning, especially of deep neural networks to be used more frequently in future marketing related services. The full potential of deep neural networks may be realized as soon as modeling starts to include highly fused input data, containing a numerical data, images and further relevant multi-media content.

References

1. Bishop, C. M., "Neural Networks for Pattern Recognition", Oxford University Press 1995
2. Bishop, C. M., "Pattern Recognition and Machine Learning", Springer 2006
3. Baydin, A.G., Pearlmutter, B.A., Radul, A.A., and Siskind, J.M., "Automatic differrentiation in machine learning: a survey", The Journal of Machine Learning Research, Vol.18, nr.153, pp1-43, 2018
4. Bevan, A., "*Machine Learning with Tensorflow*", Slides of an applied lecture, https://pprc.qmul.ac.uk/~bevan/statistics/TensorFlow_tutorial.pdf (accessed at 20.5.2018)
5. Bourez, C., "*Deep Learning with Theano*", Packt Publishing 2017
6. Chollet, F., "*Introduction to Keras*", March 9th, 2018, Slides of an applied lecture, https://web.stanford.edu/class/cs20si/lectures/march9guestlecture.pdf
7. Hinton, G.E., Osindero, S., and The, Y., "A fast learning algorithm for deep belief nets", Neural Computation, Vol. 18, pp1527-1554, 2006
8. G. Montavon, G., Orr, G.B., and Müller K.-R., "*Neural Networks: Tricks of the Trade*", Springer, 2012.
9. Gareth, J., Witten, D., Hastie, T., and Tibshirani, R., "*An Introduction to Statistical Learning",* New York, Springer, 2015.
10. Harrison, D. and Rubinfeld, D.L. "Hedonic prices and the demand for clean air", Journal of Environmental. Economics & Management, vol.5, 81-102, 1978.
11. Hirasawa, K., Wang, X., Murata, J., Hu, J., and Jin, C., "Universal learning network and its application to chaos control", Neural Networks 13, pp239-252, 2000
12. Hosmer, D. W., and Lemeshow, S., "Applied Logistic Regression". Wiley 2000.
13. Ian Goodfellow, I., Bengio, Y., and Courville, A., "*Deep Learning",* MIT Press 2016.
14. Ioffe, S., and Szegedy, C., "*Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*", 2015.
15. Kaminski, B., Jakubczyk, M., and Szufel, P., "*A framework for sensitivity analysis of decision trees*". Central European Journal of Operations Research, 2017.
16. Moro, S., Cortez, P. and Rita, P., "*A Data-Driven Approach to Predict the Success of Bank Telemarketing*". Decision Support Systems, Vol. 62, pp22-31, 2014.
17. Liou, C.-Y., Cheng, W,-C., Liou, J.-W., and Liou, D.-R., "*Autoencoder for words*", Neurocomputing, 139, 2014.

18. McLachlan, G. J., “Discriminant Analysis and Statistical Pattern Recognition”. Wiley Interscience, 2014
19. Srivastava, N., Hinton, G., Krizhevsky, A., and Sutskever, I., “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. Journal of Machine Learning Research 15 1929-1958. (2014).
20. Kotsiantis, S., Kanellopoulos, D., and Pintelas, P., "Data Preprocessing for Supervised Learning", International Journal of Computer Science, Vol. 1 N. 2, pp 111–117. (2006)
21. Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P., “Numerical Recipes”, 3rd edition, Cambridge University Press 2007
22. Schebesch, K.B., “Marktprognosen und Produktdesign mit datenorientierter KI”, Habilitation, University of Bremen 2003, 688pp
23. Trippa, L., Waldron, L., and Huttenhower, C.; Parmigiani, "*Bayesian nonparametric cross-study validation of prediction methods*". The Annals of Applied Statistics. 9 (1): 402–428. (2015).
24. Zimmermann H.G., and Weigend, A.S. “Representing Dynamical Systems in Feedforward Networks: A Six Layer Architecture”, in: Weigend, S.A., Abu-Mostafa, y., and Refenes, A.P.N. (eds.) Decision Technologies for Financial Engineering, Neural Networks in the Capital Markets 96, pp289-306, 1997
25. Repo1, http://lib.stat.cmu.edu/datasets/boston (accessed at 04.06.2018)
26. Repo2, https://archive.ics.uci.edu/ml/datasets/Bank+Marketing (accessed at 06.06.2018)
27. Rcran, https://cran.r-project.org (R version 3.5.0 accessed at 07.05.2018)