Insurance: an R-Program to Model Insurance Data

Marcos Marin-Galiano, Andreas Christmann
Department of Statistics, University of Dortmund, Germany
marcos.marin-galiano@uni-dortmund.de
christmann@statistik.uni-dortmund.de

Abstract

Data sets from car insurance companies often have a high-dimensional complex dependency structure. The use of classical statistical methods such as generalized linear models or Tweedie's compound Poisson model can yield problems in this case. Christmann (2004) proposed a general approach to model the pure premium by exploiting characteristic features of such data sets. In this paper we describe a program to use this approach based on a combination of multinomial logistic regression and $\varepsilon-$support vector regression from modern statistical machine learning.

Keywords: Claim size, insurance tariff, logistic regression, statistical machine learning, support vector regression.

1 Introduction

Insurance companies generally collect a huge amount of data from their customers. The responses of main interest are the expected claim amount per year and the probability of at least one claim. Both quantities may depend on a number of explanatory variables. Among these explanatory variables there is information about the customer and the main driver of the car, other users of the car, previous claims, description of the car, driving distance within one year, and geographical region. E.g. the value of a tariff often increases with increasing values of the yearly driving distance. In general there is a high-dimensional complex dependency structure among the explanatory variables and also between the explanatory variables and both response variables. Some of the relationships are not even monotone.

Generalized linear models are often used to construct insurance tariffs, see McCullagh and Nelder (1989), Kruse (1997) or Walter (1998). Tweedie's compound Poisson model, see Smyth and Jørgensen (2002), is more flexible than the generalized linear model because it contains not only a regression model for the expectation of the response variable, say Y, but also a regression model for the dispersion of Y. However, even this more general model can still yield problems to model complex high-dimensional relationships. E.g. if many explanatory variables are discrete with more than two possible values, which is quite common for insurance data sets, the number of all possible higher order interaction terms increases exponentially fast. Modern techniques from statistical machine learning can be helpful in this respect because they are non-parametric methods and have shown flexible behaviour in many applications.

Chapados et al. (2002) and Christmann (2004) proposed to estimate the pure premium in an indirect manner by using the law of total probability. The goal is to get additional information about the data which is not visible by a straightforward estimation of the pure premium. As a result of the law of total probability one has to estimate conditional probabilities and conditional expectations. In principle, this can be done by quite different techniques. As an example, Christmann (2004) used the pair kernel logistic regression and $\varepsilon-$support vector regression. However, even the fast software myKLR (Rüping, 2003) is computer-intensive to estimate the conditional probabilities by kernel logistic regression. Here we therefore use classical logistic regression to estimate these quantities.

The paper is organized as follows. In Section 2, we shortly outline the method proposed in Christmann (2004). In Section 3, we present two programs written in R (R, 2004) which were developed to fit and to predict the model. We mainly describe the programs and explain the program options. A typical program call is given. The main use of our programs is for medium sized data sets because of memory management in R. The conditional probabilities are fitted by classical logistic regression. Vapnik's $\varepsilon-$support vector regression (Vapnik, 1998) is used to fit the conditional expectations. Section 4 contains a discussion. The programs are listed in the Appendix.

2 Model

Insurance companies are interested to determine the actual premium (gross premium) charged to the customer. In principle, the actual premium is the sum of the pure premium plus safety loadings plus administrative costs plus the desired profit.

In this paper the focus will only be on the *pure premium*. Let n be the number of customers. For each customer i denote the number of claims during a year by n_i, the claim size of claim number j by $y_{i,j}$, and the number of days under risk by t_i, $1 \leq i \leq n$. For each customer compute

$$y_i = \frac{\sum_{j=1}^{n_i} y_{i,j}}{t_i/360} \,, \quad 1 \leq i \leq n \,,$$

which we call the individual claim amount per year. The corresponding random variables are denoted by N_i, $Y_{i,j}$, T_i, and Y_i. We assume here that $Y_{i,j} \geq 0$ almost surely. The vector of explanatory variables is denoted by $x_i \in \mathbb{R}^p$. Sometimes we omit the index i.

Our *primary response* is the pure premium

$$\mathrm{E}(Y|X = x) \,, \quad x \in \mathbb{R}^p. \tag{1}$$

The *secondary response* is the conditional probability

$$\mathrm{P}(Y > 0|X = x) \,, \quad x \in \mathbb{R}^p \,, \tag{2}$$

that the policy holder will have at least one claim within one year given the information contained in the explanatory variables. Of course, we do not have full knowledge of the conditional distribution of $Y|X = x$.

Car insurance data often show the following characteristic features, cf. Chapados et al. (2002) and Christmann (2004).

- Most of the policy holders have no claim within a year.
- The claim sizes are extremely skewed to the right, but there is an atom at zero.
- The variability differs for low and high claims.
- Extreme high claim amounts are rare events, but contribute enormously to the total sum.
- There are only imprecise values available for some explanatory variables.
- Some claim sizes are only estimates.
- The data sets to be analyzed are huge.
- There is a complex high-dimensional dependency structure between variables.

Our statistical objectives are twofold. Our goal is to model the pure premium and the probability of at least one claim, but we also like to extract additional information

often hidden in such data sets. This information can be hard to detect by estimating the pure premium in a direct manner with classical statistical methods.

The estimated pure premium should be as precise as possible. There are different criteria to measure the goodness-of-fit, e.g. the mean squared error $E((Y-\hat{Y})^2|X = x)$ or a Pearson-χ^2-type statistic. Further, an insurance tariff should be unbiased or at least approximately unbiased, i.e. $E(Y-\hat{Y}|X=x) \approx 0$. This should be valid not only for the whole population, but also for reasonable sub-populations. It is particularly difficult to implement both criterions at once.

The following model tries to exploit the characteristic features mentioned before of such insurance data sets. Both responses are modelled simultaneously.

To deal with the different variability of individual claim amount values in sub-populations, we split the data in $k+1$ classes. We do this by defining a class variable C, which describes sub-populations of customers which are interesting to the company. An example is

$$C = \begin{cases} 0, & \text{if } Y = 0 \\ 1, & \text{if } Y \in (0, 2000] \\ 2, & \text{if } Y \in (2000, 10000] \\ 3, & \text{if } Y \in (10000, 50000] \\ 4, & \text{if } Y > 50000\,, \end{cases} \tag{3}$$

see Christmann (2004). Of course, it depends on the application how many classes should be used and how reasonable boundaries can be defined. We will not address this problem here.

We decompose the pure premium $E(Y|X=x)$ with respect to C by using the law of total probability, which yields

$$E(Y|X=x) = \sum_{c=0}^{k} P(C=c|X=x) \cdot E(Y|X=x, C=c). \tag{4}$$

The first summand in (4) is equal to zero because $E(Y|X=x, C=0) \equiv 0$ and we obtain

$$E(Y|X=x) = \sum_{c=1}^{k} P(C=c|X=x) \cdot E(Y|X=x, C=c). \tag{5}$$

Note that the second response given in (2) is equal to $1 - P(C=0|X=x)$. Therefore, the model has the ability to simultaneously estimate both responses we are interested in. Furthermore, a reduction of computational time is possible because

there is no need to compute $E(Y|X = x, C = 0)$. Often more than 90% of the customers have no claim during a year. Hence it is only necessary to use less than 10% of the observations to fit the conditional expectations.

If consistent estimators of the conditional probabilities and conditional expectations in (5) are used, the pure premium is also estimated in a consistent manner (in probability or almost sure) which follows from Slutzky's theorem. This is often sufficient for practical purposes, because data sets from car insurance companies are usually large. However, this estimating approach may give biased estimates for small data sets and then a bias correction should be performed.

There are many methods to estimate the conditional probabilities and conditional expectations. Some possible combinations are:

- multinomial logistic regression and gamma regression,
- multinomial logistic regression and ϵ-support vector regression,
- kernel logistic regression and ϵ-support vector regression,
- classification trees and semi-parametric regression,
- neural networks,
- robust logistic regression and regression trees.

These choices guarantee a high flexibility to fit the model. It is even possible to estimate the conditional expectations with different regression methods. This can be of interest especially for our class $C = 4$ for the extreme values, say for y_i above 50000 EUR. This class usually contains much less observations than the other classes such that it can be favourable to use only a few explanatory variables to model this class to avoid over-fitting. There is a huge literature for modelling extreme values, *cf.* Embrechts et al. (1997), Beirlant et al. (2002), Celebrián et al. (2003) and Teugels (2003). E.g. one can model large claims with generalized Pareto distributions for the main nodes of a tree.

3 The Programs

In this paper we concentrate on one combination to fit the terms in the proposed strategy given in (5), namely the combination of multinomial logistic regression and ϵ-support vector regression (ϵ-SVR). The theory of these two methods is omitted here. We refer to McCullagh and Nelder (1989) for information about multinomial

logistic regression and Vapnik (1998), Schölkopf and Smola (2002) and Hastie et al. (2001) for information about ϵ-SVR.

Multinomial logistic regression is a standard method for estimating conditional probabilities, allows continuous and discrete explanatory variables and is implemented in many statistical software packages. Further, the algorithms used for multinomial logistic regression are much faster for large data sets than the more computer-intensive algorithms for kernel logistic regression, cf. Keerthi et al. (2002) and myKLR (Rüping, 2003). We use the ϵ-SVR for estimating the conditional expectations. This technique is very flexible − especially if used in combination with an RBF-kernel − in fitting regression models with complex dependency structures and usually gives good predictions. Our program uses a suggestion of Cherkassky and Ma (2004) in order to find a good choice of the hyperparameters for the ϵ-SVR. We take these parameters as starting points for an algorithm to optimize the choice of these hyperparameters. We use the simplex algorithm proposed by Nelder and Mead (1965). The mean squared error (MSE), mean absolute deviation and a Pearson-type statistic can be used as goodness-of-fit criterion.

The implementation was done in R (R, 2004). The program `insurance` fits the model described in Section 2 for medium sized data sets. Predictions of the pure premium for new data using a model output produced by `insurance` can be obtained by the program `predict.insurance`. The programs are available from the authors. The software codes are listed in the Appendix.

3.1 The `insurance` program

The procedure to fit the model is the following.

1. Input: Data frame which contains both training and validation data.
2. Erase all observations which have at least one missing value in one of the desired explanatory variables.
3. Construct a class variable C.
4. Randomly separate training from validation data using a random seed.
5. Run `multinom` from the R-package MASS, cf. Venables and Ripley (2002), on the training data treating all explanatory variables as discrete variables. If interaction terms for modelling the conditional probabilities are needed, the user has to specify them as explanatory variables in advance.
6. Apply the resulting model on the validation data to get the class probabilities for all data points.

7. Scale all explanatory variables to [0,1].
8. Separate the training data into data sets with the same value of the class variable.
9. Compute the suggestions for the hyperparameters proposed by Cherkassky and Ma (2004) for each class separately out of the training data.
10. Optimize the hyperparameters by using an optimizing algorithm with a deviation measure as optimization criterion. The model is trained using the training data and the deviation measure is calculated by applying the trained model to the validation data. It is possible to choose between optimizing the hyperparameters for all classes separately or for all classes at once. The latter means that a premium is estimated using the probability structure obtained in steps 5 and 6.
11. Fit the model for the optimal hyperparameters.
12. Write the fitted model into a list.

The program carries out all of these steps. It is necessary to make sure that the R-packages `e1071` (Leisch et al., 2003), `MASS` and `nnet` (Venables and Ripley, 2002) are loaded. The call of the program is

```
insurance(tvdat,varia,y,interval,seed=0,bigmean=FALSE,bigprob=FALSE,
prob.method="multinom",pred.method="svm",multiteration=300,
tratio=0.5,noiter=500,tolerance=10^(-6),nooptim=FALSE,tablet=TRUE,
criterion="MSE",pareto=FALSE,onlydata="no",opti.method="Nelder-Mead",
opti.mode="all-in-one",NA.hold=FALSE)
```

The following arguments are required:

- `tvdat`: R data frame or matrix containing the training and validation data. Observations have to be in the rows, variables have to be in the columns.
- `varia`: Vector containing the column numbers of the explanatory variables.
- `y`: Number containing the column number of the dependent variable (i.e. the claim size).
- `interval`: Vector containing the limits of the classes for defining the class variable C. The definition of the class variable in (1) is obtained by `interval = c(2000,10000,50000)`.

There are many optional arguments which can be adjusted by the user.

- `seed`: Number describing the random seed, which is used to separate `tvdat` into training and validation data. This way, it is possible to invoke the same separation into training and validation data for different choices of the variables and C in order to compare the results (default value: 0)
- `bigmean`: If `TRUE`, the expectation of $C = k$ will be the mean of all values for the claim size in this class, if `FALSE`, the expectation will be calculated by the method specified in `pred.method` (default value: FALSE).
- `bigprob`: The analogy to `bigmean` for the probability of $C = k$. If `TRUE`, the probability will be the relative frequency of the values in this class (default value: FALSE).
- `prob.method`, `pred.method`: Place holder for the estimation methods of the expectation and probability component of the model. We plan to implement some more methods in future releases of this program (default values: "multinom" and "svm").
- `multiteration`: Maximum number of iterations `multinom` will perform (default value: 300).
- `tratio`: Percentage of `tvdata` which should be used to form the training data. Must be a number between 0 and 1 (default value: 0.5).
- `noiter`: Maximum number of iterations `optim` will perform in order to find the optimal hyperparameters for the ϵ-SVR. (default value: 500)
- `tolerance`: Relative tolerance as used in `optim` as stopping criterion (default value: 10^(-6)).
- `nooptim`: If `TRUE`, the program will not optimize the parameter suggestions by Cherkassky and Ma (2004), but use these suggestions directly to fit the model (default value: FALSE).
- `tablet`: The program uses by default a frequency table in order to speed up `multinom`. However, if the number of explanatory variables is very high, it is sometimes faster to use `multinom` directly from the data without computing a frequency table in advance. In these cases, choose `tablet = FALSE` (default value: TRUE).
- `criterion`: The deviation criterion used for `optim`. Implemented so far: `"MSE"` as the standard Mean Squared Error, `"trim.MSE"` as the trimmed version in which the observations of $C = k$ are omitted. With `"meanresid"` the program uses the mean of the residuals and `"hosmer"` uses a chi-square type statistic similar to the one used in the Hosmer-Lemeshow test, cf. Hosmer and Lemeshow (1989) (default value: "MSE").
- `pareto`: Only for the case, if `bigmean=TRUE`. If `TRUE`, the expectation of the class $C = k$ is computed as the expectation of a generalized Pareto-distribution which covers the data for this class best (default value: FALSE).

- `onlydata`: This option offers the possibility to output the separated training and data set. The options `"training data"` and `"validation data"` have to be chosen for this purpose. In these cases, the output of the function is the desired data set. No optimization or fitting will be performed. The option `"no"` supplies no data sets. For more details, see the description of the output objects below (default value: "no").
- `opti.method`: The optimization algorithm used in `optim`. Every algorithm `optim` is providing can be chosen (default value: "Nelder-Mead").
- `opti.mode`: If "single", the parameter optimization will be done for each of the k estimation components separately. If "all-in-one", the optimization of all parameters for all classes will be done at once by optimizing the deviation criterion for the estimated pure premium (default value: "all-in-one").
- `NA.hold`: If FALSE, all observations with at least one NA, coded by "-9", in one of the variables, will be erased from the data set before the fitting of the model starts. If TRUE, all NA´s will remain in the data set (default value: FALSE).

If `onlydata="training data"` or `onlydata="validation data"`, the output will only consist of the specified data set in matrix form. If `onlydata="no"`, the output is a list containing the following elements:

- `pred.method, bigprob, prob.method, interval, bigmean, criterion, opti.mode, NA.hold`: Repetition of the settings made in the call.
- `pred.model`: List containing the summary for the expectation components of the model. As long as `"svm"` is the only method available at this time, the summaries equal a summary of the `svm`-function in package `e1071`. Each SVR-summary is accessible per `object$pred.model[[j]]` with j denoting the class, for which this summary holds true.
- `pbig`: The relative frequency of data in the training data set, which belongs to the highest class of claim sizes (extreme cases). It replaces the estimation of the probability of the highest class by using `multinom`. If `bigprob = TRUE`, this will be `NULL`.
- `pmodel`: Summary of the probability component of the model. See the documentation of `multinom` for details.
- `modelmat`: Matrix containing the optimized hyperparameters C, ϵ and γ for each class of claim sizes. The parameter combinations for the classes can be found in the rows of that matrix going from low claim classes on the top to high claim classes on the bottom.

- `bigmeanvalue`: If `bigmean = TRUE`, the mean of all claim sizes in the highest claim class, or, if `pareto = TRUE` has been chosen in addition, the expectation of the generalized Pareto distribution which fits the data from the highest claim class best. If `bigmean = FALSE`, this will be `NULL`.
- `scalemat`: Matrix containing the maximum (first column) and minimum (second column) of all variables.
- `beta0`: Matrix containing the hyperparameters C, ϵ and γ by using the suggestions of Cherkassky and Ma (2004). This matrix is build up the same way as `modelmat`.

3.2 The `predict.insurance` program

After fitting the model using the `insurance` function, the next step in an analysis will be to predict the claim size for new data. First of all, we need the call of the function `scale.insurance` which prepares the data set for the use of `multinom` in the `predict.insurance` function. The two calls are

`scale.insurance(model, testdat)` and
`predict.insurance(model, testdat, classdat, y=0, test=FALSE)`.

The arguments are:

- `model`: The output list of a fitted model as delivered by `insurance`.
- `testdat`: New data the user likes to evaluate. Object has to be a data frame or a matrix and the variables must be in the same order, as they are given in the call of `insurance`.
- `classdat`: The result of the `scale.insurance` function for `testdat`.
- `y`: Vector of claim sizes from the data set. Only used, if `predict.insurance` is used to compute the MSE for a new test data set. The order of claim sizes must match the order of the observations in `testdat` (default value: 0).
- `test`: If `TRUE` and a vector for `y` is supplied, the program will compute several deviation criterions for the predicted values (default value: FALSE).

The output of `scale.insurance` is a modified version of `testdat`, in which the data of all variables will be transformed into factors insuring the proper working of `multinom`.

The output of `predict.insurance` is a list containing the following components:

- `premia`: Data frame containing the variable values of each observation of `testdat` in each row along with the estimated value for the pure premium $E(Y|X = x)$.
- `pmat`: Data frame containing the variable values of each observation of `testdat` in each row along with the estimated values for the conditional probabilities $P(C = c|X = x)$.
- `costmat`: Data frame containing the variable values of each observation of `testdat` in each row along with the estimated values for the conditional expectations $E(Y|C = c, X = x)$.
- `MSE.MSE, MSE.trim, MSE.meanresid, MSE.hosmer, MSE.bias`: If `test = FALSE`, these will be `NULL`. Otherwise, they contain the corresponding deviance measures using the known claim size provided in `y`. These are: MSE, trimmed MSE, the mean of the absolute residuals, a variant of the Hosmer-Lemeshow-statistic and the bias.

These functions can be used to predict the pure premium values for a new data set, which contains data of either x_i or x_i and y_i. In the latter case some goodness-of-fit measures are also calculated.

3.3 An Example for the Usage of these Functions

The program `simul.car`, which creates a simulated data set, can be found in the same zip-file containing the main programs, which can be downloaded from our the website, mentioned in the Appendix. The simulated data set allows to test the options of our insurance programs. In this Section, we will give a little example how to use these programs. The example works under R1.9.1 with the packages `e1071` (version V1.4-1), `nnet` (version V7.2-2) and `MASS` (version V7.2-4).

Before using the function `insurance` in this example one should load functions `age.prog`, `demo.prog`, `simul.car` and `insurance.V1`. First of all, some data, which is necessary for the simulating program, is read into memory by using the functions `age.prog` and `demo.prog`.

```
age.struct <- age.prog()
demo.struct <- demo.prog()
```

Then we are able to create a training data set and some "new" data, for which we like to estimate the pure premium. The argument used in `simul.car` denotes the sample size. The following program for a sample size of $60,000$ needs around 5 minutes on a PC with 2.67 GHz.

```
set.seed(123456)
traindat <- simul.car(60000)
newdat <- simul.car(15000)[,1:2]
```

The data set contains two explanatory variables (`Age` and `Demo`) and the dependent target variable `Claim`. As an example, we like to model the effect of both explanatory variables on the claim size. We like to choose 2000 and 10000 as class borders and would like to estimate the class probability of the class "10000 and higher" by `multinom`. Furthermore, we want to choose the SVR-hyperparameters separated by each expectation component and 75% of the training data set should be used for training, the remainder for validation. The following lines show, how to fit the model in this case and how to estimate the claim size for some new data.

```
model<-insurance(traindat,1:2,3,c(2000,10000),bigprob=TRUE,
opti.mode="single",tolerance=0.2,tratio=0.75)
classdat<-scale.insurance(model,newdat)
prediction<-predict.insurance(model,newdat,classdat)
```

Some results which are of great interest, are:

- `model$beta0`: The parameter choices according to Cherkassky and Ma (2004).
- `model$modelmat`: The optimized parameters.
- `model$pmodel`: The fitted multinomial logistic regression model for computing the class probabilities.
- `model$pred.model`: The fitted SVR models for computing the regression components.
- `prediction$pmat[1:10,]`: Estimation of $P(C|X = x)$ for the first 10 observations and $C = 1, 2, 3$.
- `prediction$costmat[1:10,]`: Estimation of $E(Y|C = c, X = x)$ for the first 10 observations and $C = 1, 2, 3$.
- `prediction$premia[1:10,]`: Estimation of $E(Y|X = x)$, i.e. the estimated claim size for the first 10 observations.

In order to visualize the results, it is possible to compute the mean of one of the results in `prediction` in dependence to one of the explanatory variables, e.g.

```
plotmat<-matrix(0,ncol=3,nrow=74)
plotmat[,1]<-17:90
for (i in 1:74) { plotmat[i,2] <-
mean(prediction$premia[which(prediction$premia[,1]==i+16),3])}
for (i in 1:74) { plotmat[i,3] <-
```

```
mean(traindat[which(traindat[,1]==i+16),3])}
plot(plotmat[,1],plotmat[,2],type="b",xlab="Age",
ylab="Predicted Claim Amount",col="red", pch=1)
points(plotmat[,1],plotmat[,3],type="p",col="black",pch=16)
title("Mean Claim Amount:  predicted (red) vs.  observed (black)")
```

Plotting other results can be done accordingly.

3.4 Usage of the Programs

We allow the use of our programs for purposes of scientific research. Any commercial use must be authorized by the authors in advance. The programs can be downloaded from the website mentioned in the Appendix. Please note that we do not give any warranty for the programs and the results. If you find any bugs, please send an email to the first author with a detailed description of the problem.

4 Discussion

In this article, we refer to a common task in the field of car insurances: the estimation of the pure premium and the probability for at least one claim within a year given the values of the explanatory variables. Such data sets often contain complex high-dimensional relationships between explanatory variables and also between explanatory variables and the responses. This can cause problems for classical statistical methods like generalized linear models. As an alternative approach we suggest to use modern machine learning methods. Such methods have shown to be capable of dealing with complex data structures in many applications, see e.g. Schölkopf and Smola (2002). In order to model both responses at once, we follow the proposal of Chapados et al. (2002) and Christmann (2004) and use the law of total probability. The latter article shows the possibilities to apply the method for real-life data from 15 German motor vehicle insurance companies. In that example, the influence of age and gender of the driver on the claim size has been examined. The results show that the method is capable of describing interactions even if they have not been modelled in advance. As a consequence one has to estimate conditional probabilities and expectations. This can be helpful to detect and to model hidden structures in the data set. Further, a reduction of computation time is possible: most customers have no claim within one year, such that their data may be omitted for modelling the conditional expectations without any loss of information.

We developed two R-programs for fitting and predicting the model; their codes can be found in the Appendix. The program uses multinomial logistic regression for estimating the conditional probabilities and ϵ-support vector regression for estimating the conditional expectations. In order to find suitable hyperparameters when using ϵ-SVR, we use the simplex algorithm proposed by Nelder and Mead (1965) with the parameter suggestions of Cherkassky and Ma (2004) as starting points. Deviation measures like mean squared error, χ^2-type statistics or the mean absolute distance can be used as optimization criteria.

It is possible to find some enhancements of our programs. First of all, only the combination of multinomial logistic regression and $\epsilon-$SVR can be carried out with our R-programs. Besides some classical statistical methods there are procedures from statistical machine learning which are of interest such as kernel logistic regression or AdaBoost. These procedures have some interesting robustness properties, see Christmann and Steinwart (2004) for classification problems. Therefore, a possible method for estimating the conditional probabilities could be the inclusion of kernel logistic regression in combination with a dual algorithm proposed by Keerthi et al. (2002). This algorithm is already used in the software myKLR developed by Rüping (2003). We did not use this approach here, because kernel logistic regression is currently not available in R.

Another open problem is the treatment of extremely high claims. If we want to model the conditional expectations of the class containing these claims, we may run into the problem that there are not enough data points to guarantee an accurate estimation of complex relationships in high dimensions. Extreme value theory may be helpful, cf. Embrechts et al. (1997), Beirlant et al. (2002), Celebrián et al. (2003) and Teugels (2003).

The investigation of different estimation techniques for the conditional probabilities and regressions can result in a better performance of the proposed model. A benchmark study to compare these combinations with classical statistical method is worthwhile.

Future research is necessary to study the effect of the interval boundaries of C and the choice of useful variable selection methods. In a future paper we will discuss the impact of different goodness-of-fit measures to be minimized by the Nelder-Mead algorithm on the computation time and on the generalization error. Finally, methods from extreme values theory can be advantageous to model the extremes.

Acknowledgements

This work was supported by the Deutsche Forschungsgesellschaft (SFB 475, "Reduction of complexity in multivariate data structures") and DoMuS (Forschungsband Modellbildung und Simulation, "Statistical Software and Algorithms") of the University of Dortmund. We are grateful to Mr. A. Wolfstein and Dr. W. Terbeck from the Verband öffentlicher Versicherer in Düsseldorf, Germany, for making available the data set and for many helpful discussions.

A The `insurance` program

In this Section we present the complete R code for the `insurance` program. The R code can be downloaded from

http://www.statistik.uni-dortmund.de/sfb475/berichte/insurance.zip

For documentation on this program please refer to Section 3.1.

```
#Insurance program
#Author: Marcos Marin-Galiano, University of Dortmund
#Mail: marcos.marin-galiano@uni-dortmund.de
#Version 1.0
#This can be downloaded from:
#http://www.statistik.uni-dortmund.de/sfb475/berichte/insurance.zip
#Only for scientific use.
#NO WARRANTY.

insurance<-function(tvdat,varia,y,interval,seed=0,bigmean=FALSE,
bigprob=FALSE,prob.method="multinom",pred.method="svm",
multiteration=300,tratio=0.5,noiter=500,tolerance=10^(-6),
nooptim=FALSE,tablet=TRUE,criterion="MSE",pareto=FALSE,onlydata="no",
opti.method="Nelder-Mead",opti.mode="all-in-one",NA.hold=FALSE)

{
#Reading all required R-packages. Contribution of an output-list.
if (opti.mode!="all-in-one" & opti.mode!="single") stop("False value
chosen for opti.mode. Value must be all-in-one or single.")
library(MASS) ; library(e1071) ; library(nnet)
output<-NULL

#Erase all data with Missing Values in one of the independent
#variables (code: -9).
```

```
if (NA.hold==TRUE) ldata<-tvdat else
{if (length(varia)==1) ldata<-tvdat[which(tvdat[,varia]!=-9),]
else ldata<-tvdat[which(apply(tvdat[,varia]!=-9,1,prod)==1),]}
rm(tvdat) ; gc(TRUE)

#Computation of the number of classes (without "0"-class)
classes<-length(interval)+1

#Determine, to which class of claim size each observation belongs
#(according to interval).
classif<-rep(classes,dim(ldata)[1])
for (i in 1:(classes-1)) classif[which(ldata[,y]<
interval[classes-i])]<-classes-i
classif[which(ldata[,y]==0)]<-0
ldata<-cbind(ldata,classif)
if (length(unique(sort(classif)))!=classes+1) stop("Empty classes
present.")
rm(classif) ; gc(TRUE)

#Building a data frame, where all useless data will be omitted.
ldata<-ldata[,c(varia,y,dim(ldata)[2])]

#Re-determination of all column numbers.
varia<-1:length(varia) ; y<-length(varia)+1
classvar<-length(varia)+2

#Seperation of training and validation data.
set.seed(seed)
seperate<-sample(1:dim(ldata)[1],round(dim(ldata)[1]*tratio))
traindat<-ldata[seperate,] ; validat<-ldata[-seperate,]
rm(seperate) ; rm(ldata) ; gc(TRUE)

#Requesting, whether progamm was only used to generate a data set
#rather than analysing anything
if (onlydata=="training data") return(traindat)
if (onlydata=="validation data") return(validat)
if (onlydata!="no") stop()

#Estimating class probabilities with multinom.
if (bigprob==FALSE) classdat<-traindat[which(traindat[,classvar]<
classes),] else classdat<-traindat
if (bigprob==FALSE) pbig<-length(which(traindat[,y]>
interval[classes-1]))/dim(traindat)[1] else pbig<-NULL
classdat[,classvar]<-as.factor(classdat[,classvar])
for (i in 1:length(varia)) classdat[,i]<-factor(classdat[,i])

if (tablet==TRUE)
{
#Running multinom by using a frequency table.
```

```
freque<-as.data.frame(table(classdat[,-y]))
multiclass<-length(unique(sort(freque[,length(varia)+1])))
freque2<-matrix(freque[,length(varia)+2],ncol=multiclass)
classdat<-freque[1:(dim(freque)[1]/multiclass),varia]
if (mode(classdat)=="numeric")
    {classdat<-as.data.frame(classdat)
     names(classdat)<-names(freque)[varia]}
delete<-which(apply(freque2,1,sum)==0)
if(length(delete)!=0)
    {freque2<-freque2[-delete,] ; classdat<-classdat[-delete,]}
if (mode(classdat)=="numeric")
    {classdat<-as.data.frame(classdat)
     names(classdat)<-names(freque)[varia]}
if (prob.method=="multinom") pmodel<-multinom(as.formula(paste
("freque2~",paste(names(classdat),collapse="+"))),classdat,
maxit=multiteration) else pmodel<-NULL
}
else
{
#Running multinom directly.
if (prob.method=="multinom") pmodel<-multinom(as.formula(paste
("classif~",paste(names(classdat)[varia],collapse="+"))),classdat,
maxit=multiteration) else pmodel<-NULL
multiclass<-NULL ; freque<-NULL ; freque2<-NULL
}
rm(classdat) ; rm(multiclass) ; rm(freque) ; rm(freque2) ; gc(TRUE)

#Applying pmodel on validation data.
classdat<-validat[,varia]
if (mode(classdat)=="numeric")
    {classdat<-as.data.frame(classdat)
     names(classdat)<-names(validat)[varia]}
for (i in 1:dim(classdat)[2]) classdat[,i]<-factor(classdat[,i])
if (prob.method=="multinom") pmat<-predict(pmodel,classdat,type=
"probs") else stop("Illegal probability model")
rm(classdat) ; gc(TRUE)

#Scaling data to [0,1] and generating the formula for the desired
#model.
scalemat<-matrix(0,nrow=length(varia),ncol=2)
for (i in 1:length(varia))
  {scalemat[i,1]<-max(traindat[,i])
   scalemat[i,2]<-min(traindat[,i])
   traindat[,i]<-(traindat[,i]-scalemat[i,2])/
                 (scalemat[i,1]-scalemat[i,2])
   validat[,i]<-(validat[,i]-scalemat[i,2])/
                (scalemat[i,1]-scalemat[i,2])}
model.form<-as.formula(paste(c(names(traindat)[y],paste(names
(traindat)[varia],collapse="+")),collapse="~"))
```

```
dimnames(scalemat)[[2]]<-c("max","min")
dimnames(scalemat)[[1]]<-names(traindat)[varia]

#Seperation of traindat into partial data for each claim class.
traindat<-traindat[traindat[,classvar]>0,]
traindat<-split(traindat[,1:y],traindat[,classvar])
if (bigmean==FALSE) eff.class<-classes else eff.class<-classes-1

#Estimating the mean value for the highest claim by computating the
#mean or by using the expectation of a generalized Pareto distribution.
if (pareto==FALSE)
{
if (bigmean==TRUE) bigmeanvalue<-mean(traindat[[classes]][,y]) else
bigmeanvalue<-NULL
}
else
{
if (bigmean==TRUE)
    {
#Two functions for estimating an appropriate generalized Pareto distribution.
    ac.logdensityParetoAS<-function(x,alpha,sigma)
    {
    if (alpha!=0) {logdensity<- -sigma-(1+alpha)*log(1+x/(alpha*
    exp(sigma)))}
    if (alpha==0) {logdensity<- -sigma-x/exp(sigma)}
    return(logdensity)
    }

    ac.MLE.GPD<-function(x,xi,beta,level=0.05)
    {
    my.data.x<-x ; my.p<-2
    assign("my.data.x",my.data.x,env=.GlobalEnv)
    # f1 is the function you want to minimize
    f1<-function(theta)
    {
    logL<- -sum(ac.logdensityParetoAS(my.data.x,theta[1],theta[2]))
    return(logL)
    }
    # Starting vector
    theta0<-matrix(c(1/xi,log(beta)),ncol=1)
    # Minimization.
    out<-optim(theta0,f1,method="Nelder-Mead",control=list
    (maxit=20000,reltol=1.0E-12))
    theta<-out$par
    xi<-1/theta[1] ; beta<-exp(theta[2])
    mean.observed<-mean(my.data.x) ; expectation<-NA
    if (xi<1) expectation<-beta/(1-xi)
    out.cov<-matrix(c(0,0,0,0),ncol=2)
    out.cov[1,1]<-(1+xi)^2 ; out.cov[1,2]<-beta*(1+xi)
```

```
    out.cov[2,1]<-out.cov[1,2] ; out.cov[2,2] <- 2*beta*beta*(1+xi)
    out.cov<-out.cov/length(x)
    out.CIxi<-c(xi-qnorm(1-level/2)*sqrt(out.cov[1,1]),
                xi+qnorm(1-level/2)*sqrt(out.cov[1,1]))
    out.CIbeta<-c(beta-qnorm(1-level/2)*sqrt(out.cov[2,2]),
                  beta+qnorm(1-level/2)*sqrt(out.cov[2,2]))
    return(out=list(xi=xi,beta=beta,CI.xi=out.CIxi,
    CI.beta=out.CIbeta,cov=out.cov,CI.level=level,
    expectation=expectation,mean.observed=mean.observed))
}

    modell<-ac.MLE.GPD(traindat[[classes]][,y]-50000,10,2)
    bigmeanvalue<-modell$expectation+50000}
    else bigmeanvalue<-NULL
}

#Computing optimal parameters (c.f. Cherkassky/Ma (2004))
beta0<-rep(0,eff.class*3)
for (i in 1:eff.class)
{
var1<-mean(traindat[[i]][,y])+3*sqrt(var(traindat[[i]][,y]))
noise<-sqrt(mean((glm(model.form,family=Gamma(link="log"),
traindat[[i]],maxit=200)$fitted-traindat[[i]][,y])^2))
noise<-max(10^(-3),noise)
n<-dim(traindat[[i]])[1] ; var2<-3*noise*sqrt(log(n)/n)
var3<-1/(2*((0.35^(1/length(varia)))^2))
beta0[((i-1)*3+1):(i*3)]<-c(var1,var2,var3)
}

if (nooptim==FALSE)
{
#Optimizing the parameters of Cherkassky/Ma by using the
#Nelder-Mead-Algorithm.
#Function for estimating MSE of validat, when traindat and beta
#are used to fit the model.
MSE.func<-function(beta,traindat,validat,model.form,pred.method,
eff.class,y,pmat,bigprob,bigmean,pbig,bigmeanvalue,criterion,
opti.mode)
{
#Fitting svm model.
beta<-exp(beta)
costmat<-matrix(0,nrow=dim(validat)[1],ncol=eff.class)
if(opti.mode=="all-in-one")
    {
    if (pred.method=="svm")
    for (i in 1:eff.class)
    {model<-svm(model.form,traindat[[i]],type="eps-regression",
    cost=beta[i*3-2],epsilon=beta[i*3-1],gamma=beta[i*3],scale=FALSE)
     costmat[,i]<-predict(model,validat)}
```

```
    else stop("Illegal prediction model")
    #Computation of the estimated premia for different settings of
    #bigprob and bigmean.
    if (bigprob==TRUE & bigmean==TRUE)
    {pmat2<-pmat[,-1] ; costmat2<-cbind(costmat,bigmeanvalue)
     premia<-apply(pmat2*costmat2,1,sum)}
    if (bigprob==TRUE & bigmean==FALSE)
    {pmat2<-pmat[,-1] ; costmat2<-costmat
     premia<-apply(pmat2*costmat2,1,sum)}
    if (bigprob==FALSE & bigmean==TRUE)
    {pmat2<-cbind(pmat[,-1]*(1-pbig),pbig)
     costmat2<-cbind(costmat,bigmeanvalue)
     premia<-apply(pmat2*costmat2,1,sum)}
    if (bigprob==FALSE & bigmean==FALSE)
    {pmat2<-cbind(pmat[,-1]*(1-pbig),pbig) ; costmat2<-costmat
     premia<-apply(pmat2*costmat2,1,sum)}
    }
else
    {
    if (pred.method=="svm")
    {model<-svm(model.form,traindat,type="eps-regression",
     cost=beta[1],epsilon=beta[2],gamma=beta[3],scale=FALSE)
     premia<-predict(model,validat)}
     else stop("Illegal prediction model")
    }

#Computation of desired MSE according to the given criterion.
if (criterion=="trim.MSE" & opti.mode!="all-in-one") criterion<-"MSE"
if (criterion=="MSE") MSE<-mean((premia-validat[,y])^2)
if (criterion=="trim.MSE")
{trim.Punkte<-which(validat[,y]<50000)
MSE<-mean((premia[trim.Punkte]-validat[trim.Punkte,y])^2)}
if (criterion=="meanresid") MSE<-mean(abs(premia-validat[,y]))
if (criterion=="hosmer")
    {
    sortmat<-cbind(validat[,y],premia)
    sortmat<-sortmat[order(sortmat[,2]),]
    n<-dim(sortmat)[1] ; blocks<-max(round(n/1000),10)
    observed<-rep(0,blocks) ; expected<-rep(0,blocks)
    border<-round(n/blocks*(0:blocks))
    for (i in 1:blocks) observed[i]<-
    mean(sortmat[(border[i]+1):border[i+1],1])
    for (j in 1:blocks) expected[j]<-
    mean(sortmat[(border[j]+1):border[j+1],2])
    MSE<-sum((observed-expected)^2/expected)}
    return(MSE)
    }
#Optimization step.
if (opti.mode=="all-in-one")
```

```
beta<-optim(log(beta0),MSE.func,method=opti.method,control=
list(maxit=noiter,reltol=tolerance),traindat=traindat,
validat=validat,model.form=model.form,pred.method=pred.method,
eff.class=eff.class,y=y,pmat=pmat,bigprob=bigprob,bigmean=bigmean,
pbig=pbig,bigmeanvalue=bigmeanvalue,criterion=criterion,
opti.mode=opti.mode)$par
else
{
beta<-rep(0,eff.class*3)
for (k in 1:eff.class) beta[((k-1)*3+1):(k*3)]<-
optim(log(beta0[((k-1)*3+1):(k*3)]),MSE.func,method="Nelder-Mead",
control=list(maxit=noiter,reltol=tolerance),traindat=traindat[[k]],
validat=validat[validat[,classvar]==k,],model.form=model.form,
pred.method=pred.method,eff.class=eff.class,y=y,pmat=pmat,
bigprob=bigprob,bigmean=bigmean,pbig=pbig,bigmeanvalue=bigmeanvalue,
criterion=criterion,opti.mode=opti.mode)$par
}

}
if (nooptim==FALSE) modelmat<-matrix(exp(beta),byrow=T,ncol=3)
else modelmat<-matrix(beta0,byrow=T,ncol=3)
dimnames(modelmat)[[2]]<-c("C","epsilon","gamma")
output<-list()
if (pred.method=="svm")
#Fitting the svm models for the optimized parameters. Writing of all
#results.
{for (i in 1:eff.class)
{output[[i]]<-svm(model.form,traindat[[i]],type="eps-regression",
cost=modelmat[i,1],epsilon=modelmat[i,2],gamma=modelmat[i,3],
scale=FALSE)
}}
beta0<-matrix(beta0,ncol=3,byrow=TRUE)
dimnames(beta0)[[2]]<-c("C","epsilon","gamma")
output<-list(pred.method=pred.method,pred.model=output,
bigprob=bigprob,pbig=pbig,prob.method=prob.method,pmodel=pmodel,
modelmat=modelmat,interval=interval,bigmean=bigmean,
bigmeanvalue=bigmeanvalue,scalemat=scalemat,beta0=beta0,
criterion=criterion,opti.mode=opti.mode, NA.hold=NA.hold)
class(output)<-c("insurance","predict")
output$call<-match.call()
return(output)
}
```

B The `predict.insurance` program

These two programs are used to predict the insurance premium for new data. The second program also calculates some measures of exactness for the test data set. For documentations please refer to the main text.

```
#scale.insurance and predict.insurance program
#Author: Marcos Marin-Galiano, University of Dortmund
#Mail: marcos.marin-galiano@uni-dortmund.de
#Version 1.0
#This can be downloaded from:
#http://www.statistik.uni-dortmund.de/sfb475/berichte/insurance.zip
#Only for scientific use.
#NO WARRANTY

scale.insurance<-function(model,testdat)
{classdat<-testdat
for (i in 1:dim(classdat)[2])
classdat[,i]<-factor(classdat[,i])
classdat}

predict.insurance<-function(model,testdat,classdat,y=0,test=FALSE)
{
library(MASS) ; library(nnet) ; library(e1071)
#Computation of P(C=c|X=x)
urtestdat<-testdat
if (model$prob.method=="multinom") pmat<-predict(model$pmodel,
classdat,type="probs") else stop("Illegal probability model")
#Computation of E(Y|C=c,X=x)
costmat<-matrix(0,nrow=dim(classdat)[1],
ncol=length(model$pred.model))
for (i in 1:dim(testdat)[2]) testdat[,i]<-(testdat[,i]-
model$scalemat[i,2])/(model$scalemat[i,1]-model$scalemat[i,2])
if (model$pred.method=="svm") for (i in 1:length(model$pred.model))
costmat[,i]<-predict(model$pred.model[[i]],testdat) else
stop("Illegal prediction model")
#Computation of E(Y|X=x), equivalent to the insurance programm
if (model$bigprob==TRUE & model$bigmean==TRUE)
{pmat2<-pmat[,-1] ; costmat2<-cbind(costmat,model$bigmeanvalue)
 premia<-apply(pmat2*costmat2,1,sum)}
if (model$bigprob==TRUE & model$bigmean==FALSE)
{pmat2<-pmat[,-1] ; costmat2<-costmat
 premia<-apply(pmat2*costmat2,1,sum)}
if (model$bigprob==FALSE & model$bigmean==TRUE)
{pmat2<-cbind(pmat[,-1]*(1-model$pbig),model$pbig)
 costmat2<-cbind(costmat,model$bigmeanvalue)
 premia<-apply(pmat2*costmat2,1,sum)}
if (model$bigprob==FALSE & model$bigmean==FALSE)
```

```
{pmat2<-cbind(pmat[,-1]*(1-model$pbig),model$pbig)
 costmat2<-costmat ; premia<-apply(pmat2*costmat2,1,sum)}
#Computation of MSE (calculating all criteria, same as in insurance)
if (test==TRUE)
{
MSE.MSE<-mean((premia-y)^2)
trim.Punkte<-which(y<50000)
MSE.trim<-mean((premia[trim.Punkte]-y[trim.Punkte])^2)
MSE.meanresid<-mean(abs(premia-y))
sortmat<-cbind(y,premia) ; sortmat<-sortmat[order(sortmat[,2]),]
n<-dim(sortmat)[1] ; blocks<-max(round(n/1000),10)
observed<-rep(0,blocks) ; expected<-rep(0,blocks)
border<-round(n/blocks*(0:blocks))
for (i in 1:blocks) observed[i]<-
mean(sortmat[(border[i]+1):border[i+1],1])
for (j in 1:blocks) expected[j]<-
mean(sortmat[(border[j]+1):border[j+1],2])
MSE.hosmer<-sum((observed-expected)^2/expected)
MSE.bias<-mean(premia-y)
}
else
{MSE.MSE<-NULL ; MSE.trim<-NULL ; MSE.meanresid<-NULL
 MSE.hosmer<-NULL ; MSE.bias<-NULL}
#Outputs
premia<-cbind(urtestdat,premia) ; pmat<-cbind(urtestdat,pmat2)
costmat<-cbind(urtestdat,costmat2)
output<-list(premia=premia,pmat=pmat,costmat=costmat,MSE.MSE=MSE.MSE,
MSE.trim=MSE.trim,MSE.meanresid=MSE.meanresid,MSE.hosmer=MSE.hosmer,
MSE.bias=MSE.bias)
output$call<-match.call()
if (test==TRUE)
{
cat("The MSE is",MSE.MSE,"\n")
cat("The trimmed MSE is",MSE.trim,"\n")
cat("The mean of the absolute deviation is",MSE.meanresid,"\n")
cat("The Hosmer-Lemeshow chi-square-like MSE is",MSE.hosmer,"\n")
cat("The bias is",MSE.bias,"\n")
}
return(output)
}
```

C Example

In this section you can find the program lines for the example in Section 3.3 as long with the program code for `simul.car`, which creates a random and not necessarily realistic data set. The only purpose of this data set is to provide a possibility of

examining the functionality of our programs.

```
#Program for simulating a not necessarily realistic data set
#for use in insurance.
#Author: Marcos Marin-Galiano
#You have to carry out the functions age.struct and
#demo.struct in advance to use this program (cf. example).
simul.car<-function(n=1000,ag=age.struct,de=demo.struct)
age<-17:90
demographic<-c((2:9)*10,(1:10)*100,2000,3000,4000,5000,7000)
randa<-c(rep(0,73),1) ; randd<-c(rep(0,22),1)
datmat<-matrix(0,nrow=n,ncol=3)
dimnames(datmat)[2]<-list(c("Age","Demo","Claim"))
for (i in 1:n)
{
datmat[i,1]<-sum(age*sample(randa))
datmat[i,2]<-sum(demographic*sample(randd))
proba<-(ag[ag[,1]==datmat[i,1],2:5]+de[de[,1]==datmat[i,2],2:5])/2
claimcl<-sample(1:4,1,prob=proba)
randc<-rep(0,4) ; randc[claimcl]<-1
randclaim<-c(0,rgamma(1,shape=4.450477,scale=246.4904),
rgamma(1,shape=4.589312,scale=905.5823),
rgamma(1,shape=4.283613,scale=4305.477))
datmat[i,3]<-sum(randc*randclaim)
}
datmat<-as.data.frame(datmat) ; return(datmat)
}

#Example
age.struct<-age.prog()
demo.struct<-demo.prog()
set.seed(123456)

#Simulating training and test data sets.
traindat<-simul.car(60000)
newdat<-simul.car(15000)[,1:2]

#Modelling and predicting
model<-insurance(traindat,1:2,3,c(2000,10000),bigprob=TRUE,opti.mode="single",
tolerance=0.2,tratio=0.75)
classdat<-scale.insurance(model,newdat)
prediction<-predict.insurance(model,newdat,classdat)

#Plotting results
plotmat<-matrix(0,ncol=3,nrow=74)
plotmat[,1]<-17:90
for (i in 1:74) plotmat[i,2]<-
mean(prediction$premia[which(prediction$premia[,1]==i+16),3])
for (i in 1:74) plotmat[i,3]<-mean(traindat[which(traindat[,1]==i+16),3])
```

```
plot(plotmat[,1],plotmat[,2],type="b",xlab="Age",ylab="Predicted Claim Amount",
     col="red", pch=1)
points(plotmat[,1],plotmat[,3],type="p",col="black",pch=16)
title("Mean Claim Amount: predicted (red) vs. observed (black)")
```

References

J. Beirlant, T. De Wet, and Y. Goegebeur. Nonparametric estimation of extreme conditional quantiles. Technical Report 2002-07, Universitair Centrum voor Statistiek, Katholieke Universiteit Leuven, 2002.

A.C. Celebrián, M. Denuit, and P. Lampert. Generalized pareto fit to the society of actuaries' large claims database. *North American Actuarial Journal*, 7:18–36, 2003.

N. Chapados, Y. Bengio, V. Vincent, J. Ghosn, C. Dugan, I. Takeuchi, and L. Meng. Estimating car insurance premia: A case study in high-dimensional data inference. *Advances in Neural Information Processing*, 14:1369–1376, 2002.

V. Cherkassky and Y. Ma. Practical selection of svm parameters and noise estimation for svm regression. *Neural Networks*, 17:113–126, 2004.

A. Christmann. An approach to model complex high-dimensional insurance data. *To appear in: Allgemeines Statistisches Archiv*, 4, 2004.

A. Christmann and I. Steinwart. On robust properties of convex risk minimization methods for pattern recognition. *Journal of Machine Learning Research*, 5:1007–1034, 2004.

P. Embrechts, C. Klüppelberg, and T. Mikosch. *Modelling Extreme Events for Insurance and Finance*. Springer, Berlin, 1997.

T. Hastie, R. Tibshirani, and J.Friedman. *The Elements of Statistical Learning. Data Mining, Inference, and Prediction*. Springer, New York, 2001.

D.W. Hosmer and W. Lemeshow. *Applied Logistic Regression*. Wiley, New York, 1989.

S.S. Keerthi, K. Duan, S.K. Shevade, and A.N. Poo. A fast dual algorithm for kernel logistic regression. URL `http://guppy.mpe.nus.edu.sg/~mpessk`. National University of Singapore. Preprint., 2002.

O. Kruse. *Modelle zur Analyse und Prognose des Schadenbedarfs in der Kraftfahrt-Haftpflichtversicherung*. Verlag Versicherungswirtschaft e.V., Karlsruhe, 1997.

F. Leisch, E. Dimitriadou, K. Hornik, D. Meyer, and A. Weingessel. R package e1071, 2003. http://cran.r-project.org.

P. McCullagh and J.A. Nelder. *Generalized Linear Models*. Chapman and Hall, London, 2^{nd} edition, 1989.

J.A. Nelder and R. Mead. A simplex method for functional minimization. *Computer Journal*, 7:308–313, 1965.

R. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2004. URL: http://www.R-project.org.

S. Rüping. myKLR - kernel logistic regression. Technical report, University of Dortmund, Department of Computer Science, 2003.

B. Schölkopf and A.J. Smola. *Learning with Kernels. Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, 2002.

G.K. Smyth and B. Jørgensen. Fitting tweedie's compound poisson model to insurance claims data: dispersion modelling. *ASTIN Bulletin*, 32:143–157, 2002.

J.L. Teugels. Reinsurance acturial aspects. Technical Report 2003-006, EURANDOM, 2003.

V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

W.N. Venables and B.D. Ripley. *Modern Applied Statistics With S*. Springer, fourth edition edition, 2002.

J.T. Walter. *Zur Anwendung von Verallgemeinerten Linearen Modellen zu Zwecken der Tarifierung in der Kraftfahrzeug-Haftpflichtversicherung*. Verlag Versicherungswirtschaft e.V., Karlsruhe, 1998.