ABSTRACT

Machine Learning techniques operate data to perform classification and prediction. They are becoming more and more popular, due (among other things) to the still increasing computational power of machines. These algorithms use a very limited set of assumptions, compared to “standard” statistical methods. However, on the contrary of such methods, inference is not the main goal of Machine Learning techniques: they focus on classification and prediction. This is the reason why these “innovative” techniques are considered as more “black box” solutions.

This paper presents a comparison on a simulated database between traditional statistical predictive modelling techniques (GLM and GAM), machine learning techniques (regression trees, bagging, random forests, boosting and neural networks) and penalized regression techniques (Lasso, Ridge and Elastic Net).
INTRODUCTION

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) which exploits data to perform classification and prediction. To do so, the machine learning algorithms use a very limited set of assumptions on the data, contrarily to more "standard" statistical methods.

For many years, Actuaries and Statisticians have used historical claims data to predict future losses. They started with basic univariate statistics, moved first to regression models and since the nineties to generalized linear models (GLM), following the availability of tools and technologies. Machine Learning should be considered as the continuation of this evolution: trying to improve the predictive power of models, solving the same problems with new methods, data and computer power available. Figure 1 depicts this evolution.

Figure 1: The path toward advanced modelling

Machine learning techniques offer significant advantages over traditional models, including the availability of various types of non-linear models which can lead to a wide range of new insights. When non-linear relationships and complex patterns are involved in the data, the complex models delivered through Machine Learning outperform most traditional methods in terms of prediction accuracy, cutting the data in a new way leading to new segmentation rules. A general presentation of the Machine Learning approach has been developed in a preceding Reacfin White Paper by Arnaud Deltour and François Ducuroir¹: “Introduction to Machine Learning techniques used in the financial industry and a practical case study”

¹ See www.reacfin.com section Publications.
Figure 2 summarizes the pros and cons of the Machine learning methods compared with the statistical predictive modelling techniques.

<table>
<thead>
<tr>
<th></th>
<th>Machine learning</th>
<th>Statistical modeling</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Limits the number of assumptions</strong></td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td><strong>Inference:</strong> Assessing the reliability of modeling assumptions</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td><strong>Prediction:</strong> ability to extrapolate future or unobserved realizations of a variable given other explanatory observations</td>
<td>+</td>
<td>-/+</td>
</tr>
<tr>
<td><strong>“Big Data”:</strong> ability to handle large sets of data both in terms of number of observations (“rows”) or variables (“columns”)</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td><strong>Human interactions:</strong> ability/need of incorporating material users ex-ante opinions (e.g. Expert Judgment)</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

Figure 2 : Comparison between Machine Learning and Statistical Predictive Modelling

This article aims at illustrating how Machine Learning methods can be applied in the context of non-life pricing with a focus on frequency modelling. A comparison with traditional methods (GLM and GAM) and penalized regression techniques (Lasso, Ridge and Elastic Net) is also performed.

**DATABASE DESCRIPTION**

For educational reasons, we present here the application of some important machine learning techniques to a simplified simulated car insurance database. Working with a simulated database is convenient as:

- We know in advance the pattern to be found in the data
- We design the simulated database such that it includes some specificities which are difficult to be captured by some algorithms
- We can have as much data as needed to apply Cross-Validation techniques

The simplified frequency database we used was sampled using a Poisson distribution function where the Poisson frequency parameter \( \lambda \) was designed as a function of two explanatory variables: Age and Power. We simulated ages and powers, and then computed the following frequencies:

\[
\lambda = a (\text{age} - b)^2 + c \text{ power} + d 1_{(\text{age} \geq 60) \cap (\text{power} \geq 50)}
\]

where \( a, b, c, d \) are positive real parameters calibrated in such a way that the range of \( \lambda \) is consistent with a frequency range.

As shown in Figure 3, the Poisson frequency surface includes a non-linear interaction between the two explanatory variables and has been chosen on purpose to « fail » standard statistical methods (as GLM) and therefore show how some machine learning methods can « fix » these issues.

Figure 3 : Poisson frequency surface
The standard Poisson estimates on our database are provided in Figure 4 and exhibit the parabolic trend in age and the linear trend in power as designed.

![Figure 4: Standard Poisson estimates for age and power](image)

**Which loss functions in the Poisson context?**

Machine Learning methods minimize a cost (or loss) function whereas statistical methods usually maximize likelihood. In Machine Learning, the cost function which is minimized depends on the context. In most cases, we can simply choose the sum of squared errors:

$$\text{Error}(\hat{y}) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2,$$

where $y_i$ is the $i^{th}$ observation and $\hat{y}_i$ is the corresponding prediction.

However, when we want to predict Poisson frequencies, it is necessary to instead consider the Poisson deviance statistics:

$$\text{Error}(\hat{\lambda}) = \sum_{i=1}^{n} \left( n_i \log \left( \frac{n_i}{\hat{\lambda}_i v_i} \right) - (n_i - \hat{\lambda}_i v_i) \right),$$

where $n_i$ is the $i^{th}$ observation (i.e. the number of claims filed by the $i^{th}$ policyholder) and $v_i$ and $\hat{\lambda}_i$ are the corresponding exposure and predicted frequency.

We will use this function in order to compare the different technique.
THE OVERFITTING PROBLEM

When modelling, one should pay attention to overfitting/lack of parsimony. Overfitting occurs when a statistical model captures random error or noise instead of the underlying relationship as illustrated in Figure 5. In case of overfitting, the goodness-of-fit indicators show a good result on the dataset used for the model calibration, but the predictive power is bad. For example, when trying to explain data variability using a set of explanatory variables, the more variables you use, the better are the $R^2$, the residual sum of square, etc.

A first improvement to basic goodness of fit indicators is to use more advanced goodness-of-fit indicators which take into account the number of parameters of the model and apply penalization, such the Akaike and Bayesian information criteria. But these solutions are not satisfying as the choice of a penalization function is arbitrary.

A better solution consists in using (cross-) validation methods. These methods use two (or several) different datasets as illustrated in Figure 6:

- A **training set** to calibrate the model,
- A **test set (or validation set)** to assess the model’s predictive ability.

Two different kinds of errors are therefore defined (see Figure 7):

- The training error is calculated by applying the model to the observations used in its calibration
- The test error is the average error that results from using the model to predict the response on the test set, which was not used in calibrating the model and therefore guarantees a more realistic measure of the prediction error.

The training error decreases with model complexity whereas the test error tends to increase when the level of model complexity creates overfitting. An optimal model complexity can then be determined.
To assess the predictive power of each model, we have split the database in two parts: a training set with the first 16,000 observations and a validation set (or test set) with the remaining 4,000 observations. Each model will be calibrated on the training set and the prediction will be assessed on the validation set. This kind of split is used to gauge the prediction ability of the model on data which were not used for the calibration phase and enables thus to avoid overfitting.

**GENERALIZED LINEAR MODELS AND GENERALIZED ADDITIVE MODELS**

Generalized Linear Models (GLM’s) are widely used for pricing in the insurance industry and are ideally suited to the analysis of non-Normal data which insurance analysts typically encounter. A GLM is used to assess and quantify the relationship between a response variable and a set of possible explanatory variables. The modelling differs from Gaussian linear models in two important respects:

- The distribution of the response is chosen from the exponential dispersion family and, thus, does not need to be Normal but may be explicitly non-Normal (e.g. Poisson, Binomial, Gamma, Inverse Gaussian)
- A transformation of the mean response is linearly related to the explanatory variables. This function \( g \) is called the link function

\[
Y = g^{-1}(\beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n) + \epsilon
\]

An extensive presentation of the GLM can be found “Actuarial Modelling of Claim Counts: Risk Classification, Credibility and Bonus-Malus Systems”.

**Application on the considered database**

Figures 8 and 9 present the results of the GLM on the considered database in comparison with the known model. We observe that GLM fail to adequately capture the interaction between age and power even when we define smaller age categories (10 years or 5 years). We could of course include an interaction between age and power in the model definition to improve the predictive accuracy. Nevertheless, doing so, we enter a priori the interaction in the GLM and it is known in advance which would never be the case in practice.
The Generalized Additive Models (GAM’s) are a generalization of the GLM where continuous variables can be included. A usually good solution to model continuous variables is indeed to use a semi-parametric approach: if we are not sure about the type of influence of the continuous variable $X$ we would prefer fitting a model with an additive score of the form

$$\text{linear part } + f(X)$$

where $f$ is left unspecified and estimated from the data.

As for the GLM, the response $Y$ probability distribution has to be any member of the Exponential Dispersion family.

The mean $\mu$ of $Y$ is linked to the nonlinear score via

$$g(\mu_i) = \beta_0 + \sum_{j=1}^{P_{\text{cat}}\beta_j x_{ij}} + \sum_{j=P_{\text{cat}}+1}^{P} f_j(x_{ij}) = \text{score}_i$$

for some smooth unspecified functions $f_j$, where $g$ is the link function.

**Application on the considered database**

Figures 10 and 11 present the results of the GAM on the considered database in comparison with the known model. We observe that GAM do not significantly improve GLM results showing the same difficulties to capture the interaction.

![Figure 10: Frequency by age with GAM](image1)

![Figure 11: Frequency by power with GAM](image2)
Pros and Cons of the GLM and GAM

The GLM usually result in a final multiplicative structure that is easily understandable by all stakeholders. There exist a lot of (commercial or free) software implementing the GLM so that it is easy to use. Moreover, GLM are based on a strong statistical framework (i.e. confidence interval, hypothesis tests,...) that helps in the fine-tuning of the model (selection of variables, grouping of modalities,...).

On the other hand, it is difficult to detect and capture interactions between variables or complex structures with GLM. Moreover, the treatment of continuous variable is not flexible.

The GAM allow to treat the continuous variables (e.g. for adequate categorization of the variables, zoning,...) while combining with the power of GLM under a strong statistical framework.

Nevertheless, GAM often face computational/convergence problems when used with too many variables and introduce an additional complexity (e.g. keeping a continuous variable in the tariff is not always adequate or easy to explain).

DECISION TREE

Decision trees can be used for regression as well as for classification problems. Iteratively, the whole database will be split into different buckets, which will also be split in smaller buckets, and this process will be repeated several times, until reaching a certain target.

Tree enables to segment the predictor space into a number of simple regions defined according to the covariates. Splitting rules can be summarized in a tree view. For each region the prediction is set as the region average.

At each node, 3 key questions will arise:

1. On which variable should the split be based? And what is the best splitting point for this variable?
2. How do we define this notion of “best” splitting variable/point?
3. How is it decided to stop the splitting process (i.e. when is a node considered as a leaf or not)?

The regression tree algorithm, which answers all these questions, is presented in Appendix 1.

Continuing to grow the tree until each leaf node corresponds to the lowest impurity will result in overfitting. If splitting is stopped too early, the error on the training data is not sufficiently low and predictive power will potentially suffer. In the decision tree algorithm, the strategy is to build the deepest regression tree and then go back by “pruning” the tree. To “prune” the tree, we have to figure out if a split is “valuable” or not. Pruning is out of scope of this article.
Application on the considered database

Figure 12 presents the resulting regression trees with two different rules of pruning.

![Figure 12 - Resulting regression trees](image)

Figure 13 presents the results of the regression trees on the considered database in comparison with the known model. We observe that the deepest RT seems to better take into account the interaction between the two explanatory variables.

![Figure 13 - Frequency by age and by power with regression trees](image)

Interpretation of decision trees results is straightforward and easy, thanks to a wide set of possible illustrations. However, the prediction quality can be poor and this is why alternative methods based on trees have been introduced, as bagging, random forest or boosting.
BAGGING

Bootstrap aggregation, or Bagging, is a general-purpose procedure for reducing the variance of a statistical learning method. It is frequently used in the context of decision trees. Recall that given a set of \( n \) independent observations \( Z_1, Z_2, \ldots, Z_n \) each with variance \( \sigma^2 \), the variance of the mean \( \bar{Z} \) of the observations is given by \( \frac{\sigma^2}{n} \). Averaging a set of observations therefore reduces variance. However, multiple training sets are usually not at disposal that is why the Bagging procedure allows to artificially creating them thanks to a bootstrap procedure.

The Bagging procedure can therefore be set as follows:

1. Bootstrap, by taking repeated samples from the (single) training data set. Doing so, we generate \( B \) different training data sets.
2. Train the method on each of the bootstrapped training sets, in order to get an array of \( B \) predictions corresponding to the response value \( \{ f_b(x) \} \)
3. Average all the predictions, to obtain the final prediction:

\[
\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} f_b(x)
\]

Application on the considered database

Figure 15 presents the results of the bagging procedure on the considered database in comparison with the known model. Several different numbers of bootstrapped training sets are used leading to a set of different trees which are then aggregated to get the prediction. We observe that the predictive power improves with the number of training sets used in the procedure.
Random forests provide an improvement over bagging thanks to an additional step that decorrelates the trees. This reduces the variance when we average the trees.

As in bagging, we build a number of decision trees on bootstrapped training samples. But when building these decision trees, each time a split in a tree is considered, a random selection of \( m \) predictors is chosen as split candidates from the full set of \( p \) predictors in order to reduce the dependency between the different trees. The split is allowed to use only one of those \( m \) predictors.

A fresh selection of \( m \) predictors is taken at each split (among the \( p \) predictors), and typically we choose \( m \approx \sqrt{p} \) that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors. Note that random forest with \( m = p \) is equivalent to bagging.

When increasing the number of trees, the prediction error usually decreases until reaching a certain level. When this level is reached, the corresponding number of trees is classically considered as a good choice.

**GRADIENT BOOSTING MACHINE**

Like bagging, boosting is a general approach that can be applied to many statistical learning methods used for regression. We only discuss here boosting for decision trees.

Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model. Notably, each tree is built on a bootstrap data set, independent of the other trees.

Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees. It can be described with the following algorithm:

1. Set \( \hat{f}(x) = 0 \) and \( r_i = y_i \) for all \( i \) in the training set,
2. For \( b = 1, 2, 3, \ldots, B \), repeat:
   a. Fit a tree \( \hat{f}^b \) with \( d \) splits (\( d + 1 \) terminal nodes) to the training data \((X, r)\),
   b. Update \( f \) by adding in a reduced (shrunken) version of the new tree:
      \[ \hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x), \]
   c. Update the residuals:
      \[ r_i \leftarrow r_i - \lambda \hat{f}^b(x_i), \]
3. The final prediction is provided by
   \[ \hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \]
Unlike fitting a single large decision tree to the data, which can lead to overfitting, the boosting approach instead learns slowly. At each step of the algorithm, we fit a decision tree to the residuals coming from the previous model. The additional tree is then added into the model in order to update the residuals. Usually, trees which are used in Boosting methods are rather small, with just a few terminal nodes, determined by the parameter $d$ in the algorithm.

Using small trees, enables to improve slowly $\hat{y}$ in areas where the previous model does not perform well. Thanks to the shrinkage parameter $\lambda$, the learning process can be slow down, allowing more and different shaped trees to explore the residuals.

Regarding Boosting method, different parameters have to be chosen. First, the number of trees $B$. Unlike bagging and random forests, boosting can overfit if $B$ (or $d$ see later) is too large. Cross-validation can be used to select $B$. Second, the shrinkage parameter $\lambda$ is a small positive number. It controls the rate at which boosting learns. Typical values are 0.01 or 0.001. Very small $\lambda$ can require using a very large value of $B$ in order to achieve good performance. The number of splits $d$ in each tree, which controls the complexity of the boosted ensemble, is the last parameter. Often $d = 2$ works well, in which case each tree enables to capture a first level of interaction. More generally, $d$ is the interaction depth, and controls the interaction order of the boosted model, since $d$ splits can involve at most $d$ variables.

**Application on the considered database**

Figure 18 presents the results of the boosting procedure on the considered database in comparison with the known model. We observe that a certain number of trees are necessary to obtain reliable results. Using a parameter $d = 2$ also improves the predictive power.

Figure 18: Frequency results with Gradient Boosting Machine
PRO’S AND CON’S OF TREE-BASED TECHNIQUES

We have completed our journey in the world of Machine Learning tree-based methods, and it is time to rapidly assess their advantages and drawbacks.

On the positive side, trees are very easy to explain to people. In simple cases, they can be displayed graphically, and are therefore easily interpretable (at least when the number of leaves is limited). Interactions and non-linear decision boundary are easily implemented with the use of these methods. They can also produce “confidence intervals” (with bagging and random forests). An interesting element is also that bagging and random forests can be parallelized (not boosting).

However, one has to use advanced techniques in order to improve the prediction quality. When the number of splits and leaves is very large, the resulting trees are not easily understandable. Moreover, these methods restrict classification to rectangular regions. Overfitting can also be observed if model is not carefully built (e.g. pruning and cross-validation). Moreover, a global optimum is not guaranteed.

In conclusion, aggregating many decision trees (using bagging, random forest or boosting) can substantially improve the predictive performance of trees but at the expense of some loss of interpretation.

NEURAL NETWORK

Neural networks (NN) are often used to perform pattern recognition (unsupervised learning). They can learn on their own and adapt to changing conditions (based on the data). NN are inspired by biological nervous systems, as e.g. human brain’s information processing mechanism: they are composed of a large number of interconnected processing elements (neurons) working together to solve problems.

The neurons of the human brain collect signals from other neurons through dendrites. They then send electrical activity through axon which splits in thousands of branches. At both ends of each branch, a synapse converts the activity of the axon into electrical effects that excite activity in the connected neurons.

As the real neurological neuron, an artificial neuron is an element with several inputs and one output, as depicted in Figure 19. It has two modes of operation. In training mode (calibration), the neuron can be trained to fire (or not), depending on the inputs. For example, in pattern recognition, we associate outputs with input patterns. In using mode (prediction), the neuron uses calibrated values to fire (or not), depending on the inputs. We can e.g. identify input pattern and try to output the associated output pattern.

There exist two important types of neurons: perceptrons and sigmoid neurons.
Network representation: connected neurons with different layers

There are three layers of neurons: the input layer, the hidden layer(s) and the output layer, as illustrated in figures 20 and 21. The input layer (left) contains the input neurons. The hidden layer(s) (middle) is called “hidden” as neurons in this layer are neither inputs nor outputs. The choice of the number of layers/neurons can be determined by cross-validation. One hidden layer is sufficient for most of the problems, but additional layers can be added if it increases the performance (networks with 2 or more layers are called deep Neural Networks). The output layer (right) contains the output neurons.

Two different types of Neural Networks exist. The most used ones are feed-forward networks, with only one way: from input to output.

In recurrent/feedback networks (which are less commonly used), loops are allowed.

Two categories of learning methods

Learning methods can be classified in supervised learning and unsupervised learning.

In supervised learning, for each input in the training dataset, a response variable is known and a NN is used to fit this response. The goal is to be able to predict the response for inputs which are not in the training dataset. Cross-validation is thus important in order to avoid overfitting (bias-variance tradeoff). The NN learns from its error and can adapt to perform the best fit: various algorithms exist to modify the weights and/or threshold. The mostly used is the Backpropagation algorithm, which is discussed in more details in the appendix 2 of this document.

In unsupervised learning, no response variable is given in the training dataset. NN is then used to construct clusters of similar patterns.
Application on the considered database

Figure 22 presents the results of the neural networks on the considered database in comparison with the known model. Several NN have been tested: 1 hidden layer with 1 neuron, 1 hidden layer with 8 neurons and 2 hidden layers with 2 neurons.

Figure 23 shows the resulting neural networks in the 2 cases we have tested.

Figure 22 : Frequency results with Neural Network

Figure 23 : Resulting neural networks (left: 1 hidden layer with 8 neurons, right: 2 hidden layers with 2 neurons)

Pros and cons of the neural networks

The neural networks are easy to conceptualize and flexible in the type of data they can support. There is a large amount of academic research and lots of libraries / implementations available to obtain quickly a solution. Moreover, Neural Networks can be treated probabilistically (e.g. Bayesian neural networks).

On the other side, there are alternatives that can be simpler, faster, easier to train, and provide better performance (SVM, decision trees, regression). Multi-layer neural networks are usually hard to train and opaque so that you don’t really understand how your network is solving the problem. Finally a global optimum is not guaranteed with Neural Networks.

In conclusion, Neural Networks are not magic and are not a substitute for understanding the problem deeply.
PENALIZED REGRESSION TECHNIQUES

When the number (n) of data points is far larger than the number (p) of explanatory variables (n >> p), Maximum Likelihood Estimates (MLE) have low bias and low variance provided the GLM is correctly specified. Thus, the model performs well on test observations.

However, if p becomes large with respect to n, there can be a lot of variability in the (MLE) fit. It leads to overfitting and we observe poor predictions on future observations not used in model training.

Constraining, or shrinking the estimated regression coefficients often substantially reduces the variance at the cost of a negligible increase in bias. This leads to substantial improvements in prediction accuracy for observations not used in model training.

New methodologies have been introduced that penalize the likelihood: the objective function becomes

- Log-likelihood + penalty

The penalty constrains the coefficient estimates, shrinking them towards 0. We present two main methods, Lasso and Ridge, that have different penalties applied to the likelihood:

- Penalty = \( \lambda \sum_{j=1}^{p} |\beta_j| \) for LASSO (Least Absolute Selection and Shrinkage Operator)
- Penalty = \( \lambda \sum_{j=1}^{p} \beta_j^2 \) for Ridge

\( \lambda \) is a tuning parameter (that can be determined by Cross-Validation). As \( \lambda \) increases, the flexibility of the coefficients decreases, leading to decreased variance but increased bias. For small values of \( \lambda \), the variance often decreases very rapidly, with little increase in bias.

LASSO

In Lasso, the minimization problem with constraint is the following:

\[
\min_{\beta} [- \log \text{likelihood}] \quad \text{subject to } \sum_{j=1}^{p} |\beta_j| \leq t
\]

As depicted in Figure 24, Lasso regression shrinks the coefficient towards 0. The penalty \( L^1 \) has the effect of forcing some of the coefficient estimates to be exactly equal to 0 when the tuning parameter \( \lambda \) is sufficiently large.

As \( \lambda \) increases, more coefficient estimates are set to zero (so less covariates enter the model). Hence, the Lasso also performs variable selection and generates models much easier to interpret than those produced by other shrinkage methods (e.g. Ridge regression).
Ridge

In Ridge, the minimization problem with constraint is the following:

$$\min_{\beta} \left[ -\log \text{likelihood} \right]$$

subject to $$\sum_{j=1}^{p} \beta_j^2 \leq t$$

As depicted in Figure 25, Ridge regression also shrinks the coefficient towards 0. But in the case of the Ridge regression, the penalty $$L^2$$ does not have the effect of forcing some of the coefficient estimates to be exactly equal to 0, even when the tuning parameter $$\lambda$$ becomes large.

Ridge penalty shrinks the coefficients of correlated predictors towards each other whereas the Lasso tends to pick one of them and discard the others. Hence, the Ridge regression does not perform variable selection. Models produced by Ridge regression tend to be less easy to interpret than those produced by other Lasso method.

Elastic Net

As presented in Figure 26, Elastic Net combines Lasso and Ridge penalty. It consists in Minimizing –log-likelihood with penalty mixing $$L^1$$ and $$L^2$$:

$$\min_{\beta} \left[ -\log \text{likelihood} + \lambda \left( (1 - \alpha) \sum_{j=1}^{p} \beta_j^2 + \alpha \sum_{j=1}^{p} |\beta_j| \right) \right]$$

The minimization problem with constraint is therefore the following:

$$\min_{\beta} \left[ -\log \text{likelihood} \right]$$

subject to $$(1 - \alpha) \sum_{j=1}^{p} \beta_j^2 + \alpha \sum_{j=1}^{p} |\beta_j| \leq t$$
Application on the considered database

Figure 27 presents the results of the penalized regression techniques (Lasso, Ridge and Elastic Net) on the considered database in comparison with the known model.

Figure 28 illustrates the shrinking of the coefficients towards 0 (for the 3 methods) and towards each other in the Ridge method. We clearly observe that the Elastic Net method is a mix of Lasso and Ridge.
Pros and Cons of the penalized regression techniques

The techniques are usually better than the usual methods of automatic selection (forward, backward, stepwise) and Lasso is a good feature selection tool.

Nevertheless, it might lose some relevant independent variables along the way (e.g. Lasso penalization tends to randomly select one in a group of highly correlated variables). Moreover, for $p>n$ problems, Lasso allows only to have $n$ non-zero coefficients.

In conclusion, Elastic Net which combines Lasso and Ridge penalization is usually a better feature selection solution.
**COMPARISON OF THE RESULTS**

Figure 29 summarizes the prediction error for all tested methods:

- **AVG**: Poisson global average estimator;
- **GLM1 / GLM2 / GLM3 / GLM4**: Generalized Linear Model without age classes / with 10 years age classes / with 5 years age classes / with 1 year age classes;
- **GAM1**: Generalized Additive Model;
- **RT1 / RT2 / RT3**: Regression Tree without complexity specification / with 1 SD rule / with min SD rule;
- **BAG1 / BAG2 / BAG3**: Bagging with default parameters / improved parameters / super-improved parameters (e.g. nb of iterations);
- **GB1 / GB2 / GB3**: Gradient Boosting with 100 trees / 10000 trees / 100000 trees and improved parameters;
- **NN1 / NN2 / NN3**: Neural Network with (1) hidden cell / (8) hidden cells / (2)(2) hidden cells;
- **RL1**: Ridge regression;
- **RL2**: LASSO regression;
- **RL3**: Elastic net.

We can conclude that no model overperforms all the other ones with respect to all the indicators. Neural Networks, Bagging and Boosting seem to provide the smallest prediction error in this case but we have not optimized all the parameters of each model for this basic example. Therefore, data must lead the modelling process, and the more reliable and accurate method should be determined with care, depending on the database. We therefore advise to compare several models on the database to be analyzed and to select the final model taking also into account its purpose and use (e.g. should it be transparent? Who are the final users?...).
Company profile

Reacfin is a consulting firm focused on setting up top quality, tailor-made Risk Management Frameworks and offering state-of-the-art actuarial and financial techniques, methodologies and risk strategies. We focus primarily on serving Financial Institutions.

Developments in finance and actuarial techniques are progressing at a rapid pace. Reacfin assigns highly skilled and experienced practitioners employing advanced analytics and complex predictive models. Our support, which is strongly rooted in the spirit of co-development, allows for an effective transfer of knowledge such that your firm will achieve sustainably improved performance and new competitive advantages.

Reacfin is a spin-off of the University of Louvain (which ranks 1st globally for Master degrees in Actuarial Sciences for the 3rd consecutive year according to EdUniversal). We maintain strong links with the academic world which allows us to stay current and guarantees the independence, robustness and appropriateness of the advice and services we offer.

Practical deliverables

Leveraging on our unique combination of academic expertise and practitioners experience, Reacfin delivers financial institutions with pragmatic solutions designed for real-life use while deploying state-of-the-art technical innovations.

Tools & Model developments

Tailored computational solutions designed and developed to smoothly integrate into your company’s systems and processes (incl. open-source offerings)

Advisory

Model deployments, calibration, documentation, testing, validation, maintenance and life-cycle management

Training

Both On-Site and On-Line executive education solutions including theoretical and methodological concepts, real-life case studies, exercises
FURTHER READING

General topics


Particular topics

- Y. Freund and R. Schapire,"A decision-theoretic generalization of on-line learning and an application to boosting", in *Journal of computer and system sciences*, vol. 55, n° 1, 1997, p. 119-139.
APPENDIX 1: THE REGRESSION TREE ALGORITHM

The regression tree algorithm can be summarized as follows:

- Define a loss/error (or objective) function,
- Try to find regions $R_1, R_2, \ldots, R_k$ that minimize (or maximize) the function retained,
- All possible regions definitions can of course not be considered,
- The tree algorithm therefore:
  - Starts with the global population and find the optimal split of the predictor at that level using the entire population,
  - The same process is then applied on each sub-population.

It is important to remark that the division decision is done in function of information available at moment before division execution and that there is not warranty that the division decision taken is the best alternative insight to future divisions.

At each split, one has to find the variable according to which split must be done and, moreover, determining which is the cutting value. This can be achieved through the “Impurity of a node” concept which is explained hereafter.

For a covariate $X_j$, define splitting point $s$ which split the dataset into $R_1(j, s)$ and $R_2(j, s)$:

$$
\begin{align*}
R_1(j, s) &= \{x| x_j \leq s\}, \\
R_2(j, s) &= \{x| x_j > s\},
\end{align*}
$$

where $R_1$ and $R_2$ are the two children nodes of the parent node $T$.

Regression trees use the segmentation strategy called “divide and conquer”, taking as goodness of split criterion an error measure. Specifically, if $E(T)$ is the Error measure or Loss function of set $T$ (or node in regression tree terminology), then the goal is to minimize following expression:

$$E(T) = E(R_1) + E(R_2).$$

Usually for the regression case, the residual sum of square is used as loss function, meaning that the function to minimize is equal to the sum of the square differences between the observations and the mean responses of the regions:

$$
\sum_{i : x_i \in R_1(j, s)} (y_i - \bar{y}_{R_1})^2 + \sum_{i : x_i \in R_2(j, s)} (y_i - \bar{y}_{R_2})^2,
$$

where $\bar{y}_{R_i}$ represents the mean response for the training observations in $R_i(j, s)$.

Reminder: for Poisson we use deviance statistics instead of the sum of square:

$$E(\hat{\lambda}) = \sum_{i = 1}^{n} \left( n_i \log \left( \frac{n_i}{\hat{\lambda}_i v_i} \right) - (n_i - \hat{\lambda}_i v_i) \right),$$

where $n_i$ is the $i^{th}$ observation and $v_i$ and $\hat{\lambda}_i$ are the corresponding exposure and predicted frequency.
As discussed supra, an important point is to determine when to stop the splitting process and thus to determine the number of leaves. This is linked to the bias-variance tradeoff. If a small number of splits are performed, the variance is very low but the bias will in general be quite high as only a few buckets were created. On the contrary, if many splits are performed, the bias will be very low (overfitting problem) but the variance will be big, meaning that a small change related to the data (remove one data, add one or a small change in one variable) will lead to very different prediction. The overfitting problem can be summarized as follows.

The splitting process is repeated until each leaf node corresponds to the lowest impurity or when some minimum node size (i.e. a specific minimum number of observations) is reached. Continuing to grow the tree until each leaf node corresponds to the lowest impurity will result in overfitting: the noise is followed too closely instead of the underlying relationship and it produces good predictions on the training set, but poor test set performance.

However, if splitting is stopped too early, the error on the training data is not sufficiently low and predictive power will potentially suffer. In the decision tree algorithm, the strategy is to build the deepest regression tree $T_{max}$ and then go back by “pruning” the tree. We have to figure out if a split is “valuable” or not.

**APPENDIX 2: DESCRIPTION OF THE NN CALIBRATION PROCESS**

We present here the calibration process which is usually used for NN. The very high number of parameters involved in these methods makes this step difficult. The goal is to find the weights and the thresholds such that the outputs will fit the data in the most adequate way. The cost function to be minimized will be the sum of the error square roots

$$C(W, T) \equiv \frac{1}{2N} \sum_x \|y(x) - a(x)\|^2,$$

where $N$ represents the total number of training inputs, $y(x)$ is the observed output of training data $x$ and $a(x)$ is the output given by the network for training data $x$.

Gradient descent algorithm can be used to find the optimal weights and thresholds. To do so, backpropagation algorithm is needed to compute the partial derivatives needed for the gradient descent algorithm. Several iterations are required until the method converges and the minimum value is then obtained (issues regarding convergence and local minima are out of the scope of this paper).

For the sake of simplicity, suppose the goal is to minimize a function $f(z_1, z_2)$. The variation $\Delta f$ of $f$ following a variation $\Delta z_1$ of $z_1$ and $\Delta z_2$ of $z_2$ is given by

$$\Delta f \approx \frac{\partial f}{\partial z_1} \Delta z_1 + \frac{\partial f}{\partial z_2} \Delta z_2.$$

The goal is to find variations $\Delta z_1$ and $\Delta z_2$ to make $\Delta f$ negative. Let us denote the gradient vector $\nabla f = \left(\frac{\partial f}{\partial z_1}, \frac{\partial f}{\partial z_2}\right)^T$. So, clearly, with $\Delta z = (\Delta z_1, \Delta z_2)^T$, the variation of $f$ becomes $\Delta f \approx \nabla f \cdot \Delta z$. Taking $\Delta z = -\alpha \nabla f$ with $\alpha$ being a small positive parameter (learning rate) will lead to a decrease of the function $f$ as $\Delta f$ will be negative: $\Delta f = -\alpha \|\nabla f\|^2$. 
In the case of the Neural Networks, derivatives of the cost function with respect to the weights and the thresholds have to be computed in order to find the optimal parameters (i.e. weights and thresholds). However, computing these partial derivatives \( \frac{\partial C}{\partial w_{jk}} \) and \( \frac{\partial C}{\partial \tau_j} \) (with \( w_{jk} \) being the weight connection from \( k^{th} \) neuron layer \( (l - 1) \) and \( j^{th} \) neuron layer \( l \) and \( \tau_j \) being threshold neuron \( j \), layer \( l \) ) is not straightforward. In order to do so, we have to use the backpropagation algorithm. It allows computing these derivatives recursively:

1. Begin the computation at the highest layer (output layer),
2. From the results of layer \( i \) : compute the values at layer \( i - 1 \),
3. Repeat the procedure until getting the results for all layers.

One all the derivatives are known, the Gradient descent method can be applied, and the objective function can then be minimized, leading to calibrated values for the parameters.
Machine Learning applications to non-life pricing
Frequency modelling: An educational case study

Reacfin’s 4 Centers of Excellence

<table>
<thead>
<tr>
<th>Non-Life</th>
<th>ALM, Portfolio Management &amp; Quantitative Finance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pricing with advanced predictive models:</strong></td>
<td><strong>Development of ALM models:</strong></td>
</tr>
<tr>
<td>▪ Machine learning (Regression Tree, Random forest, Gradient Boosting Methods, Neural networks...)</td>
<td>▪ Computing advanced indicators designed for financial reporting, evolved dashboards and business decision support (including market &amp; biometric risks, ALM interactions, clients behaviour, management actions, regulations)</td>
</tr>
<tr>
<td>▪ Statistical models (Generalized Linear models, Generalized Additive Models, Generalized Linear Mixed Models)</td>
<td><strong>Design of Capital Management solutions:</strong></td>
</tr>
<tr>
<td>▪ Advanced Credibility techniques and Bonus Malus methods for experience rating</td>
<td>▪ Securitization, hedging, diagnostic, compliance with regulations</td>
</tr>
<tr>
<td><strong>Reserving:</strong></td>
<td><strong>Development of Valuation tools:</strong></td>
</tr>
<tr>
<td>▪ Development of Traditional triangles methods</td>
<td>▪ For financial instruments and derivatives (multiple pricing methods (including simulation-based stochastic models), Greeks &amp; sensitivities)</td>
</tr>
<tr>
<td>▪ Implementation of Advanced individual claims reserving models</td>
<td><strong>Development of ‘Real World’ tools &amp; solutions:</strong></td>
</tr>
<tr>
<td><strong>DFA models:</strong></td>
<td>▪ For Portfolio Management, Trading and Active Risk Management (simulation-based stochastic models, stress tests, time series used for SAA, KRI’s &amp;KPI’s dashboards)</td>
</tr>
<tr>
<td>▪ Development of stochastic cash-flow projection models,</td>
<td><strong>Operations, restructuring and qualitative risk management:</strong></td>
</tr>
<tr>
<td>▪ Model calibration</td>
<td>▪ Mergers &amp; Acquisition</td>
</tr>
<tr>
<td>▪ Implementation of model testing framework</td>
<td>▪ Processes optimization</td>
</tr>
<tr>
<td>▪ Reinsurance Modelling &amp; optimisation</td>
<td>▪ Instituting Qualitative Risk Management Frameworks</td>
</tr>
<tr>
<td><strong>Life, Health and Pension</strong></td>
<td><strong>Definition of tailor-made risk strategies, risk policies and practical risk guidelines</strong></td>
</tr>
<tr>
<td>▪ Development of cutting edge Assets &amp; Liabilities Models for life insurance, including advanced modeling of management actions and customer behavior.</td>
<td><strong>Development of the Risk Appetite Framework, including risk appetite levels, tolerance margin and risk limits</strong></td>
</tr>
<tr>
<td>▪ Expertise in various valuation frameworks (Solvency 2, behaviourCEV ...) and in the computation and optimization of capital requirements (Solvency 2, ICA).</td>
<td><strong>Advising on Risk Management Governance &amp; Organization</strong></td>
</tr>
<tr>
<td>▪ Extensive knowledge of pricing and reserving methods for life and health insurance.</td>
<td><strong>Definition and implementation of ORSA Framework &amp; Internal Model Validation (incl. Designing &amp; deploying the Validation Framework for qualitative and quantitative risks)</strong></td>
</tr>
<tr>
<td>▪ Advanced validation framework for the technical provisions (Solvency 1 and 2).</td>
<td><strong>Policies embedding</strong></td>
</tr>
<tr>
<td>▪ Tools for advanced modeling of prospective mortality and adverse selection in mortality</td>
<td><strong>Our Partners</strong></td>
</tr>
</tbody>
</table>

Professor Pierre Devolder
Chairman of the Board
Pierre.Devolder@reacfin.com

François Ducuroir
Managing Partner
Francois.Ducuroir@reacfin.com

Maciej Sterzynski
Managing Partner
Maciej.Sterzynski@reacfin.com
Machine Learning applications to non-life pricing

Frequency modelling: An educational case study

Check our online resources on www.reacfin.com

Online Apps & Tools demo’s

Latest research & programs of trainings

Latest news & publications

Reacfin Apps

Check our website and online apps on www.Reacfin.com
About Reacfin

Reacfin is a consulting firm focused on setting up top quality, tailor-made Risk Management Frameworks and offering state-of-the-art actuarial and financial techniques, methodologies and risk strategies. We focus primarily on serving Financial Institutions.

Developments in finance and actuarial techniques are progressing at a rapid pace. Reacfin assigns highly skilled and experienced practitioners employing advanced analytics and complex predictive models. Our support, which is strongly rooted in the spirit of co-development, allows for an effective transfer of knowledge such that our clients achieve sustainably improved performance and new competitive advantages.

Reacfin is a spin-off of the University of Louvain (which ranks 1st globally for Master degrees in Actuarial Sciences for the 3rd consecutive year according to EdUniversal). We maintain strong links with the academic world which allows us to stay current and guarantees the independence, robustness and appropriateness of the advice and services we offer.

Reacfin s.a./n.v.
Place de l’Université 25
B-1348 Louvain-la-Neuve
Belgium

Phone: +32 (0)10 68 86 07
info@reacfin.com
www.reacfin.com