Dr. Nelvis Fornasin und Attila Gulyás

Reserving for Workers' Compensation claims using Machine Learning & Natural Language Processing

A case study from a Kaggle competition

Introduction

Machine learning advances at an unprecedented pace these days. ML algorithms can read traffic lights and are expected to drive cars soon. From an actuarial perspective, the next step is necessarily that they adequately reserve for accidents they might cause.

Jokes aside, the question of how and how much such algorithms can be integrated into the reserving processes of insurance companies is very real and urgent. Companies making use of these new solutions might gain an edge over their competitors, while at the same time laying themselves open to critics, for example, for lack of transparency.

The "Actuarial Loss Prediction" (ALP) competition aimed to further this discussion by creating a reserving challenge that goes beyond conventional methods. The challenge was hosted on Kaggle, the leading platform for data science competitions with a community of over 8 million data scientists, and was presented as follows (see [1]):

„The Actuaries Institute of Australia, Institute and Faculty of Actuaries and the Singapore Actuarial Society are delighted to host the Actuarial loss prediction competition 2020/21 to promote development of data analytics talent especially among actuaries. The challenge is to predict Workers Compensation claims using highly realistic synthetic data.

The data is fully synthetic and not specific to any legal jurisdiction or country. We are grateful to Colin Priest for building and supplying the dataset.

We invite the competitors to take claims inflation into account."

The ALP ran from December 2020 to April 2021, keeping 140 teams of actuaries and data scientists busy. At the end we retained the second place and some good insights about how machine learning in reserving might look like in the future, which we would like to share in this article.

Aim of the competition

The introduction of the ALP already provides a succinct description of the aim: "The challenge is to predict Workers Compensation claims using highly realistic synthetic data.". In this section, we provide a bit more detail on the aim of the competition and the scoring metric.

The 90.000 rows dataset was divided into a train set (54.000 rows) and a test set (36.000 rows). The test data was further split equally into a public and a private test set. The former was publicly available and was used to score the models throughout the competition. The latter was retained by the competition host and only used once at the end of the competition for the final evaluation of the models. The predictions were scored according to the root mean squared error (RMSE).

Data Exploration

The train dataset contained the fields below. The test dataset contained the same fields except for the explained variable *UltimateIncurredClaimCost*.

- **ClaimNumber:** Unique policy identifier
- **DateTimeOfAccident:** Date and time of accident
- **DateReported:** Date the accident was reported
- **Age:** Age of the worker
- **Gender:** Gender of the worker
- **MaritalStatus:** Martial status of the worker: (M)arried, (S)ingle, (U)nknown
- **DependentChildren:** The number of dependent children
- **DependentsOther:** The number of dependants excluding children
- **WeeklyWages:** Total weekly wage
- **PartTimeFullTime:** Whether the worker was employed (P)art time or (F)ull time
- **HoursWorkedPerWeek:** Total hours worked per week
- **DaysWorkedPerWeek:** Number of days worked per week
- **ClaimDescription:** Free text description of the claim
- **InitialIncurredClaimCost:** Initial estimate by the insurer of the claim cost
- **UltimateIncurredClaimCost:** Total claims payments by the insurance company

Some records in the train dataset were erroneous. For instance, for 37 records the value of HoursWorkedPerWeek was greater than 168 hours. In 122 rows the WeeklyWages was 1, a value that seemed unrealistic considering other values of the column. These data points were either removed or recoded based on expert judgment.

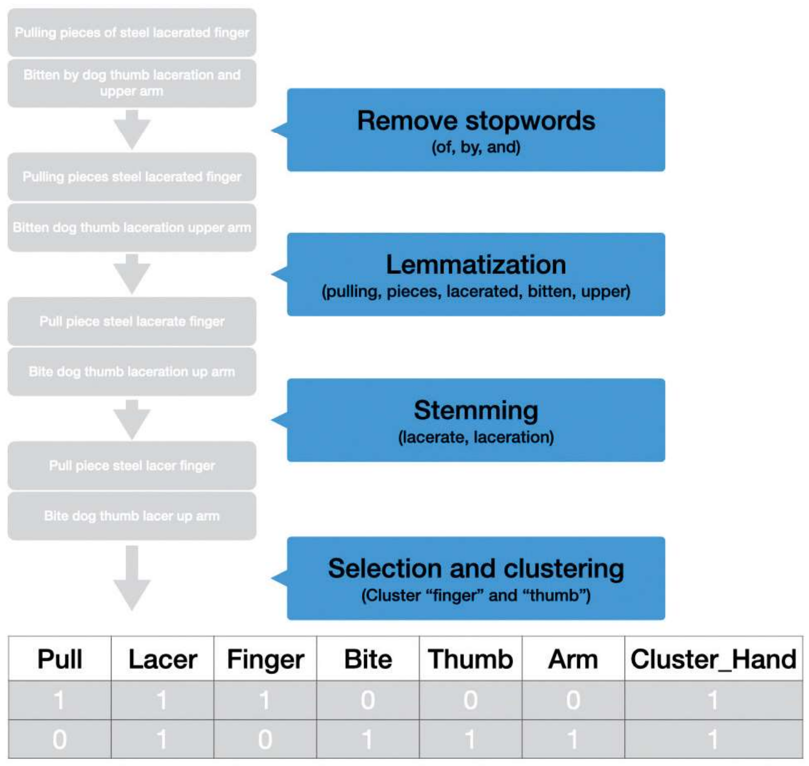
Some records in the train dataset were implausible due to extreme values of either the dependent variable or the independent variables. If we were unable to ascertain the reason for the extreme value based on the available information, we removed the record from the dataset. One cannot expect a model to predict an extreme value, if one itself cannot explain it. For instance, the largest ultimate claim was 4 million – about 5 times higher than the second largest ultimate claim –, but none of the explanatory variables suggested such a high claim.

It may have been possible to account for those extreme data points via the evaluation metric, had the metric been free to choose. However, the evaluation metric could not be changed and was by nature sensitive to prediction errors that are large in absolute terms. Therefore, the best decision seemed to be the removal of those few extreme data points.

Some new features were added, e.g. whether the accident happened within core working hours. Date(time) features were also used to create new variables, such as reporting delay. The explanatory variables were partly transformed to alleviate or reinforce the effect of extreme data points. For instance, the *InitialIncurredClaimCost* was transformed with the natural logarithm function.

Large claims often pose a problem in non-life reserving. Therefore, it is common practice to model them separately. Although there are well-established techniques for large claim modelling, there is no clear guidance on how to decide whether a claim is large. Accordingly, we tested different large claim thresholds and trained models for those claims separately. Our attempts in this regard were futile. The explicit large claim modelling did not lead to significant model improvements. Therefore, instead of training different models depending on the size of the *UltimateIncurredClaimCost*, we combined models – trained on the whole dataset – based on their ability to account for large claims. For more information, please see the Backtesting section below.

Figure 1:
NLP workflow


Pulling pieces of steel lacerated finger
Bitten by dog thumb laceration and upper arm
Remove stopwords
(of, by, and)
Pulling pieces steel lacerated finger
Bitten dog thumb laceration upper arm
Lemmatization
(pulling, pieces, lacerated, bitten, upper)
Pull piece steel lacerate finger
Bite dog thumb laceration up arm
Stemming
(lacerate, laceration)
Pull piece steel lacer finger
Bite dog thumb lacer up arm
Selection and clustering
(Cluster "finger" and "thumb")

Pull	Lacer	Finger	Bite	Thumb	Arm	Cluster_Hand
1	1	1	0	0	0	1
0	1	0	1	1	1	1

Natural Language Processing

The chapter is titled Natural Language Processing (NLP) because the *ClaimDescription* was analyzed in part with NLP techniques. NLP techniques range from overarching sentiment analysis, which describes the overall emotional content of a piece of text, to syntactic analysis, which breaks the text in sentences and tokens to be analyzed singularly (e.g. Google Cloud's natural language API, see [2]).

Unfortunately, the full potential of NLP techniques could not be exploited because of two reasons. First, the descriptions were sometimes too short, ruling out e.g. sentiment analysis. Second, for data protection reasons some descriptions had been distorted to a point where they became hardly intelligible, e.g. "TO RIGHT LEG RIGHT KNEE". Therefore, simple NLP was combined with text analysis to extract information from the claim descriptions.

The descriptions were preprocessed as follows: First, stopwords were removed to reduce the noise in the data. The set of stopwords were taken from the nltk corpus ([3]). Second, the words were lemmatized using SpaCy ([4]), as well as stemmed using the SnowBall stemmer of nltk. Lemmatization reduces the word to its form that appears in the dictionary, stemming reduces the word to its root form. The former produces semantically meaningful word forms, whereas the latter may not. For example, the lemma of lacerated is lacerate, the stem form of lacerated, laceration or lacerate is lacer.

Next, we introduced 3 categories: body part, type of wound, and accident cause. For each category we defined a list of words based on the dataset as well as our own judgment. The lists contained words like: ankle, knee, eye (body parts), strain, bruise, fracture (types of wound), slip, explosion, spider (accident cause). Moreover, we grouped these words based on semantic similarity to allow for potential inconsistencies in the labeling. For instance, face, cheek and jaw were clustered together, the rationale behind being that a face injury may be very similar to an injury of the cheek or jaw.

For each word in the lists, a new column was created and added to the dataset. The columns contained the number of occurrences of the word in the claim descriptions. For each cluster, a new column was created as well. The value of these columns was 1 if any of the words in the cluster appeared at least once in the claim description, otherwise 0. All these numeric columns representing the claim descriptions were given as inputs to the model.

The workflow is summarized in Figure 1 for two claim descriptions: Clustering and stemming make sure that the model can detect the common aspect of the two descriptions, i.e. a laceration on the claimant's hand, while differentiating its cause (respectively pulling something and a bite) and retaining more specific information.

Modelling with xgboost

We implemented our model in Python using the xgboost package. In the words of its own authors, this library "implements machine learning algorithms under the Gradient Boosting framework". In this section, we briefly refresh the concepts of random forest and gradient boosting, and then describe the way xgboost introduces probability distributions in its algorithms.

Random Forests and Gradient Boosting

Let us recapitulate the concepts of random forests and gradient boosting:

- **Random forests** are made up of decision trees – hence the "forest" –, each of which is only allowed to train on a randomly selected subset of the training set – hence the "random". Each of the decision trees populating the forest predicts a result, and a unique result is then provided by taking the average of all results (for regression problems) or the most common one (for classification problems).
- **Gradient boosting** is a machine learning technique in which the same algorithm is applied iteratively. After a first prediction has been provided, we fit a new model on its residuals - this is the first boosting round. We can then combine the first two models and model their residuals, which would be the second boosting round, and so forth.

In combining random forests and gradient boosting we modify the classic decision tree algorithm in two different ways. The predictions improve at the cost of interpretability: building up a forest we lose the possibility of looking at the individual splits of our decision tree, and each boosting round adds a new forest. A simple way of visualizing feature importance is counting how many times a variable is split on. More important variables tend to be selected more often for growing the tree.

Regression to distributions in xgboost

Suppose that we want to predict a numerical outcome Y based on a set of n dependent $X_1,\dots,X_n$, and that we suspect Y to be gamma distributed with mean μ: a common way to approach this regression problem would be to set up a generalized linear model (GLM) around the equation
$g(\mu) = \log(\mu) = \beta X$
where $\beta = (\beta_1, \dots, \beta_n)$ is a vector of regression parameters to be determined and we used „log" as link function. We can determine the βs by maximizing likelihood.

This approach cannot be translated directly to decision trees, which describe the outcome Y in terms of piecewise constant functions rather than polynomials. However, it is possible to introduce the likelihood in the computation via the loss function.

The learning process of a machine learning algorithm is guided by minimizing a loss function. Given a vector of true values Y and of model outcomes $\hat{Y}$ we can, for instance, ask for the $\hat{Y}$ that minimizes the squared error $(\hat{Y} - Y)^2$ or the squared log error $\log\left(\frac{\hat{Y}+1}{Y+1}\right)^2$.

One typical issue of the squared error is that it is driven by large outcomes – a small percentual error in predicting a large value, possibly due to noise, has a great impact on the squared error. This effect is tamed by the squared log error. So we can choose different loss functions to target different peculiarities in the data and guide the learning process of the algorithm.

Loss functions are in principle fully customizable, but xgboost already provides a wide selection, including the negative of the log likelihood to gamma and tweedie distributions with the „log" link function. Minimizing the negative of the likelihood obviously maximizes the likelihood; this is, however, not achieved by finding the optimal betas (as in GLMs), but by finding the best splits for each decision tree.

In practice, for a Gamma distribution with mean μ and shape parameter k we have the density function

$$f(x;k,\mu) = \frac{x^{k-1}e^{-\frac{xk}{\mu}}}{\Gamma(k)\left(\frac{\mu}{k}\right)^k}$$

We can set the *objective* parameter of xgboost to *reg:gamma* to obtain the negative log likelihood as loss function, see Formula 1.

Here k is interpreted as a weight parameter and defaults to 1. Also notice that in this computation we took for

Formula 1

$$L(\hat{Y},Y;k) = \sum_{i=1}^{n}\left[\log\Gamma(k_i) + \frac{Y_i k_i}{\hat{Y}_i} - k_i \log\left(\frac{Y_i k_i}{\hat{Y}_i}\right) - (k_i - 1)\log(Y_i)\right]$$

simplicity the identity as link function. For details, please see the source code of xgboost ([5]) at and the discussion at [6].

Model analysis

An xgboost model can be quite complex. These are some of the parameters that need to be set, with some sample values:

```
'objective':'reg:gamma',
'eval_metric':'rmse',
'eta': 5e-2,
'max_depth': 4,
'min_child_weight': 6,
'subsample': 0.7,
'num_parallel_trees': 50
'alpha' : 2e+2
```

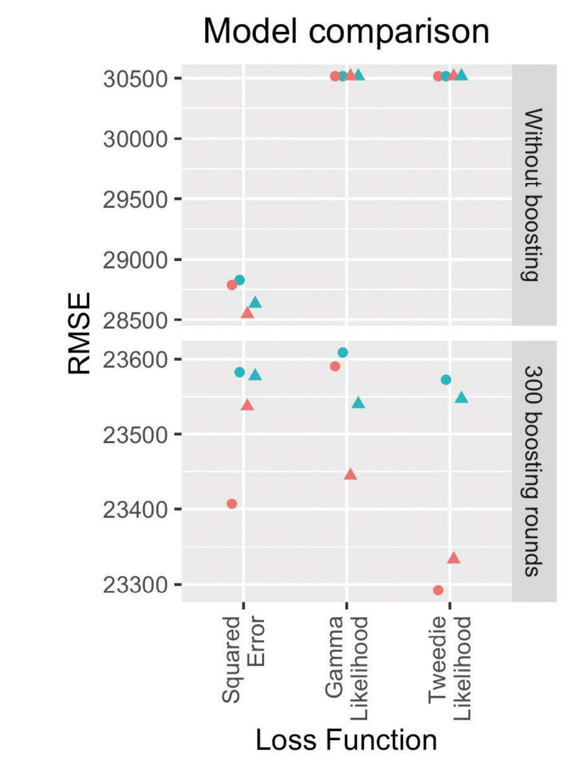
For a description and complete list of parameters, please see the official xgboost documentation ([7]).

We selected the hyperparameters for our model by specifying a hyperparameter space and looking for the best combination inside that space by randomly picking some of the points, training the respective model and choosing the one with best RMSE.

Note that this does not guarantee that the best combination will always be found: it is important for the search space to be both large enough to cover different configurations and fine enough lest we miss good configurations by accident. In practice, determining the right space is mostly a trial and error process, where it is helpful to start with a rather large grid and then zoom in on the minima: if we start off with a max depth interval between 1 and 10 and we mostly find minima around 4, it might make sense to restrict the search interval to numbers between 2 and 6. We determined the initial hyperparameter space following current best practices.

Once the parameters are set, we can train a model in Python for a given number of boosting rounds as seen in Illustration 1.

Figure 2:
Model Comparison



Model comparison

It is important to understand that the hyperparameters interact with each other, and several optimal configurations might be possible. Moreover, the preprocessing might affect the choice of hyperparameters. The hyperparameter space might need to be adjusted depending on the output of the preprocessing. In other words, the preprocessing and the hyperparameter search are interdependent.

With this remark in mind, we would like to analyze which aspects played the most important role in producing good predictions. In Figure 2, we display the RMSE of the predictions on the test dataset for different choices of hyperparameters and feature engineering: we considered all possible combinations of the three loss functions (squared error, gamma distribution and tweedie distribution), NLP analysis (with vs without), boosting (300 boosting rounds vs no boosting), bagging (random forest with 50 trees vs single tree).

In Figure 2, we can see that the major improvement is brought about by boosting. It should also be noted that unboosted models do not appear to make use of the *ClaimDescription* and are not powerful enough to detect the distribution behind the data: for unboosted models the squared error is by far the best loss function, for boosted models it is outperformed by tweedie likelihood.

We also see this in Figure 3, which compares the drop in RMSE on the

Illustration 1

```
import xgboost as xgb
xgb.train(params=parameters, dtrain=xgb.DMatrix(input_data, label=y), n
um_boost_round=300)
```

test set after each boosting round for a model with squared error and tweedie likelihood loss function: this drop is much more regular for the squared error loss function, which is directly related to RMSE, and more of a side effect for the tweedie likelihood loss function. In the latter case the RMSE abruptly drops in the blue shaded area (boosting rounds 15 to 75) and then stabilizes again.

When preprocessing the data, we used feature importance as a guideline to determine how much signal was detected from each of the explanatory variables. Seeing, for example, how the weekday of the accident was deemed important by the model – hence presumably carried signal – we clustered the weekday feature into weekend/non weekend. This had the effect of both increasing the carried signal and diminishing overfitting. The amount and type of NLP clusters considered were also largely guided by merging irrelevant clusters with similar median ultimate and breaking up relevant, large clusters to see if the model could extract more signal from the single components.

Backtesting

In this section we compare the predictions of the model with the true ultimates. Unfortunately, it is not possible to make the comparison on the test data because the *UltimateIncurredClaimCost* was not accessible to the participants of the competition. Therefore, to illustrate the predictive power of the model we run the backtest on the train dataset. The results must be interpreted accordingly.

The input data for the prediction contained an initial estimation for the ultimate cost. This initial cost estimate proved to be the most powerful predictor, which underlines the importance of human intelligence in the era of machine learning. The model essentially took and boosted this initial estimation using the remaining predictors. Therefore, we included the *InitialIncurredClaimCost* in Figure 4.

Figure 3:
Boosting for different loss functions

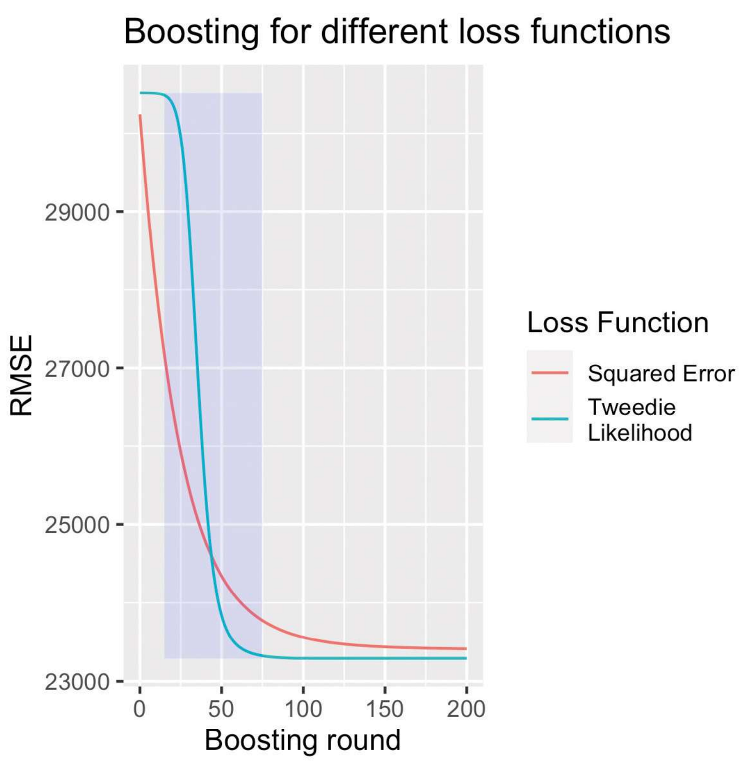


Figure 4:
Empirical probability density function of the logarithm of the initial claim cost estimate, the ultimate incurred claim cost, and the predicted incurred claim costs

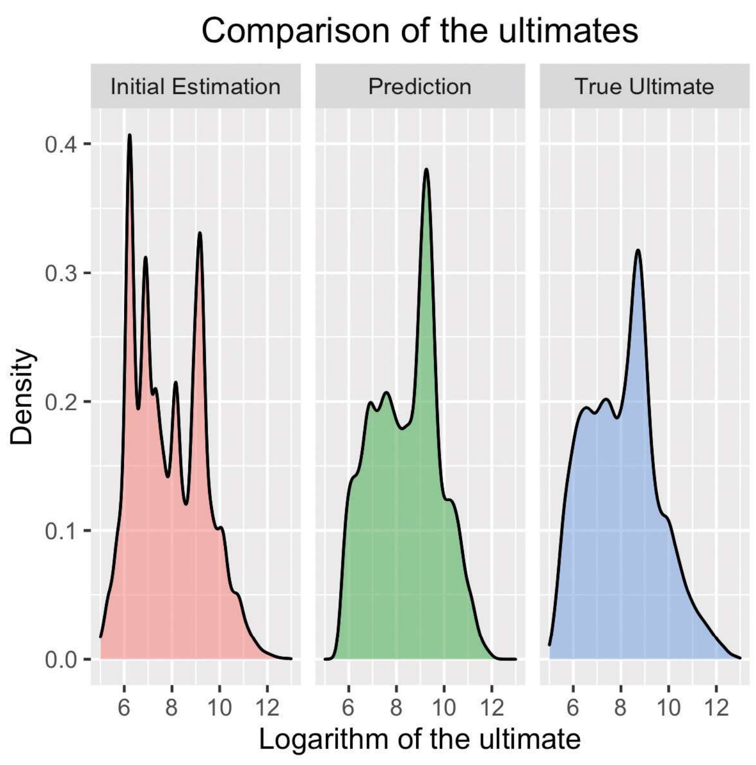
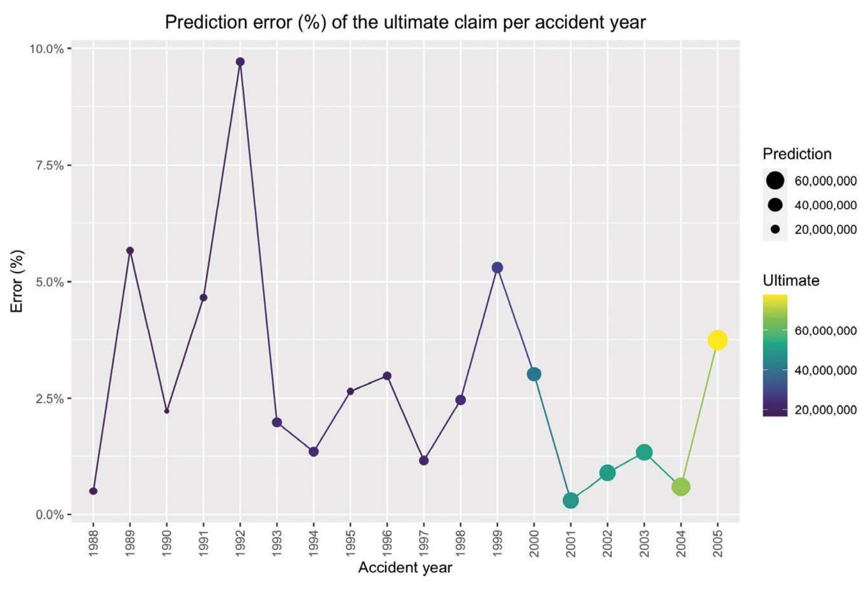


Figure 5:
Percentual error per accident year



One conundrum of this challenge was the prediction of large losses. The models struggled to effectively identify large claims, that eventually impaired the prediction accuracy of non-large claims, too. Small claims were overestimated, large claims were underestimated. Therefore, in the final submission we combined models in a way that mitigated this problem: if the mean of the predictions of a few selected models were below a certain threshold, we selected the prediction with the lowest value, otherwise the one with the highest value. As a result, the small claims were overestimated less, and the predictions for large claims improved as well.

The aim of the competition was modelling the ultimate cost at policy level. In the current practice of reserving, however, the ultimate cost is estimated at an aggregate level. Hence, it might be instructive to look at the model performance per accident year, see Figure 5

On an aggregate level, the predicted ultimate claim was 1.7% less than the true ultimate claim. True ultimates and the respective predictions grow with time – claim inflation was mentioned by the organizers of the challenge as being one of the drivers in the ultimate costs and is correctly captured by the model. But lack of interpretability in the models makes it hard to rely on the results: we do not have any direct way of explaining why the prediction for 1992 is off by roughly 10% and for 2003 by 0,3%.

In interpreting the results, it is important to remark that we did not have any additional information for old claims as compared with the new ones, i.e. information on claim development since the first estimation was not available.

Conclusion

This competition was an excellent exercise in applying machine learning to an actuarial problem. We tried several different machine learning algorithms, including neural networks, but used only xgboost for our final submission. While boosting on neural network is a known concept (see e.g. [8], thanks to M. Kiermayer for the hint), an off-the-shelf package equivalent to xgboost is not readily available, to the best of our knowledge. Therefore, powerful models may remain unexploited only because they are hard to implement with the available packages.

Interpretability of the results remained a problem throughout. A straightforward way of interpreting the model is not available yet and is perhaps just not possible, given the way ML algorithms are constructed. This remains an open challenge for the future.

In closing, we would like to thank the organizers and hosts of the competition for this valuable and fun-packed experience. Also, we would like to thank W. Abele for his useful advices and discussions, and the whole Non-life Actuarial team at Deloitte for its support.

Attila Gulyás ist Aktuar bei Deloitte Actuarial & Insurance Services in Deutschland. Er engagiert sich für Insurance Analytics in der Schaden- und Unfallversicherung. Seine aktuellen Schwerpunkte sind IFRS17 und ML-Methoden in der Tarifierung.

Dr. Nelvis Fornasin ist Senior Consultant im Non-Life Actuarial Insurance Services bei B&W Deloitte mit Schwerpunkt Actuarial Data Science. Seine beruflichen Tätigkeiten liegen u.a. im Bereich IFRS17 und Insurance Analytics.

References

[1] https://www.kaggle.com/c/actuarial-loss-estimation

[2] https://cloud.google.com/natural-language/docs/basics

[3] https://www.nltk.org

[4] https://spacy.io

[5] https://github.com/dmlc/xgboost/tree/master/src

[6] https://stats.stackexchange.com/questions/484555

[7] https://xgboost.readthedocs.io/en/latest/parameter.html

[8] https://arxiv.org/abs/2002.07971v2