Micro-reserving for Workers' Compensation claims: a case study from a Kaggle competition

Nelvis Fornasin

Machine learning is a rapidly growing field with applications in various industries, including insurance. Actuaries, in particular, are interested in understanding how machine learning models can be used to improve the reserving process and gain a competitive edge. The "Actuarial Loss Prediction" (ALP) competition, hosted on Kaggle at the beginning of 2021 by the Actuaries Institute of Australia, Institute and Faculty of Actuaries, and the Singapore Actuarial Society, provided the ideal opportunity for actuaries and data scientists to explore this topic. The competition aimed to predict Workers Compensation claims using highly realistic synthetic data. I took part in this competition with my former colleague Attila Gulyas and our team achieved the second place.

Aim of this article is to provide an overview of the competition and the dataset used, describe the machine learning model we developed and the techniques we employed, and discuss the results we achieved and the insights we gained. I hope that this article will inspire actuaries and other professionals in the insurance industry to explore the potential of machine learning in their own work.

The article will be divided in the following sections:

1. Data Exploration: this section will give an overview of the dataset used, including the variables, the outliers and the preprocessing steps.
2. Methodology: this section will discuss the machine learning model developed and the techniques used to improve its performance.
3. Results: this section will present the results achieved by the model and the evaluation metrics used.
4. Conclusion and future work: this section will summarize the main findings of the study

and suggest areas for future research.

Data Exploration

The dataset, which consisted of 90,000 rows, was split into two parts: a training set containing 54,000 rows and a test set containing 36,000 rows. The test set was further divided into two sub-sets: a public test set and a private test set. The public test set was made available for participants to evaluate their models throughout the competition, while the private test set was kept by the competition organizer and only used for the final evaluation of the models. The models were evaluated based on the root mean squared error (RMSE) of their predictions.

Each data point consisted of information related to a single claim. Next to more usual information, such as the date and time of the accident, the marital status of the workers involved and their weekly wage, the dataset also provided a claim description in free text. The most common claim descriptions are as follows:

Claim Description	# of Occurences
SLIPPED ON ROLLER TENDONITIS RIGHT SHOULDER	289
SORTING ALUMINIUM BARS STRAIN SHOULDER NECK	278
SLIPPED USING LATHE IN EYE CORNEA	258
REDBACK SPIDER BITE RIGHT FOOT RIGHT FRACTURE	256
LIFTING TYRES LOWER BACK STRAIN	255
LIFTING BACK STRAIN LOWER BACK STRAIN	253
FELL DOWN STAIRS BACK SPRAIN	236
HIT AIR HOSE LACERATED LIP	233
JAMMED RIGHT HAND PUNCTURE WOUND RIGHT ARM	231
USING AIR HOSE STRAIN RIGHT KNEE RIGHT	229

Aim of the exercise was to predict the Ultimate Claims Cost for the given claim.

Some records in the dataset were erroneous or extremely implausible, other features could be refined to help the model extract signal more clearly. Here are two examples from the preprocessing function we used to clean and refine the dataset:

```
# No more than 24*7 hours per week
data['HoursWorkedPerWeek'] =
numpy.where(data['HoursWorkedPerWeek']>=168, np.nan,
data['HoursWorkedPerWeek'])

# Weekday might be interesting
data['WeekDayOfAccident'] = data.DateTimeOfAccident.dt.weekday
```

We removed records with implausible values from the dataset if the reason for the extreme value was not clear to us based on the available information: for example, the highest ultimate claim was 4 million dollars, which was 5 times larger than the next highest claim, but none of the explanatory variables indicated such a large claim amount.

Excluding stopwords, the claim descriptions contained 3,665 different unique words. Several attempts with pre-trained NLP models proved less successful than manually dividing claim descriptions into categories based on the words they contained. These were for example the clusters corresponding to eye injuries and those where a vehicle was involved:

```
eye = ["fragment", "cornea","foreign", "corneal", "eye", "eyelid"]
motor = ["vehicle", "truck", "motor"]
```

In the next step, columns "eye" and "motor" would be added, with a value of 1 in case the claim description contained one of the words in the clusters, and 0 otherwise. Here is an example of how this would look like if we only had these two clusters:

ClaimDescription	eye	motor
DUST FLEW INTO EYE RIGHT CORNEAL ABRASION	1	0
DRIVING VEHICLE STRAINED UPPER ARM	0	1
PLAYING VOLLEYBALL STRAIN RIGHT ARM RIGHT	0	0

The complexity of the information provided rises with the number of clusters - with these two, the model can already differentiate eye injuries for cases where a vehicle was involved in the accident and where it was not. Having only a few large clusters resulted in losing information and underfitting, many small clusters lead to excessive influnce of noise and overfitting, a typical conundrum in such cases. In the end we settled on a total of 40 clusters, grouping body parts, wound types and causes of the accident.

Methodology

For the modelling part we quickly settled on XGBoost, as attempts with different methods (such as feedforward neural networks) or frameworks (such as lightGBM) consistently delivered worse results.

The code needed for training an XGB model is not very involved, this is an example:

```
import xgboost as xgb

params = {'objective':'reg:tweedie',
          'eval_metric':'rmse',
          'eta':0.13,
          'max_depth':5,
          'subsample':0.67,
          'min_child_weight':6,
          'colsample_bylevel':0.95,
          'colsample_bynode':0.75,
          'colsample_bytree':0.9,
          'gamma':2000,
          'alpha':1.6,
          'monotone_constraints': ['DependentChildren',
'HoursWorkedPerWeek'],
          'tweedie_variance_power':1.44,
          'num_parallel_tree':50}

def train_xgb_cv(config):
    data=xgb.DMatrix(train_data, label=y)
    results = xgb.cv(config=params, data, num_boost_round=200, nfold=5)
```

The challenge lies thus not in writing the code, but in deciding which values of the parameters make more sense. Some business knowledge is important here, for example in deciding that the model should maximize likelihood to a tweedie distribution ('reg:tweedie' parameter) rather than just minimizing the mean square error

, or in reasoning that more dependent children or hours worked per week should result in higher ultimate claims, everything else being equal. For some other parameter choices, say the maximal depth of the trees or the number of the trees that should be trained in parallel, business knowledge is not of great help: here we resorted to grid search. This is an example of implementation using the ray package and the function defined above:

```
from ray import tune

search_space = {
    'objective':'reg:tweedie',
    'eval_metric':'rmse',
    'eta': tune.uniform(1e-2, 2e-1),
    'max_depth': tune.randint(3, 6),
    'min_child_weight': tune.randint(4, 8),
    'subsample': tune.uniform(0.5, 0.8),
    'colsample_bytree' : tune.uniform(0.8,0.99),
    'colsample_bylevel': tune.uniform(0.8,1.0),
    'colsample_bynode' : tune.uniform(0.6,0.8),
    'gamma' : tune.loguniform(1e+1, 1e+5),
    'alpha' : tune.loguniform(1e+0, 5e+4),
    'tweedie_variance_power' : tune.uniform(1.1,1.6)
    }

analysis =
tune.run(train_xgb_cv,config=search_space,metric='rmse_mean',num_samples
=25)
```

In a way, this only brings the question one step further: how should the boundaries of the search space be defined? I do not have a satisfactory answer, and in fact no other answer than "Trial and error". It is not a question of the mathematical knowledge or programming tools currently at disposal, but rather the paradox of needing the knowledge that an ideal model would be able to provide in order to be able to define the parameters of such a model, so that if we knew the parameters then we would not actually need the model, and likewise having already the model we would not need parameters. A global and profound view of the dataset is needed in order to hope to be able to find ideal parameters, but such a view can only be won if the ideal model is already available. The current strategy of trial and error might be intellectually unexciting, but at least does not lead to paradoxes.

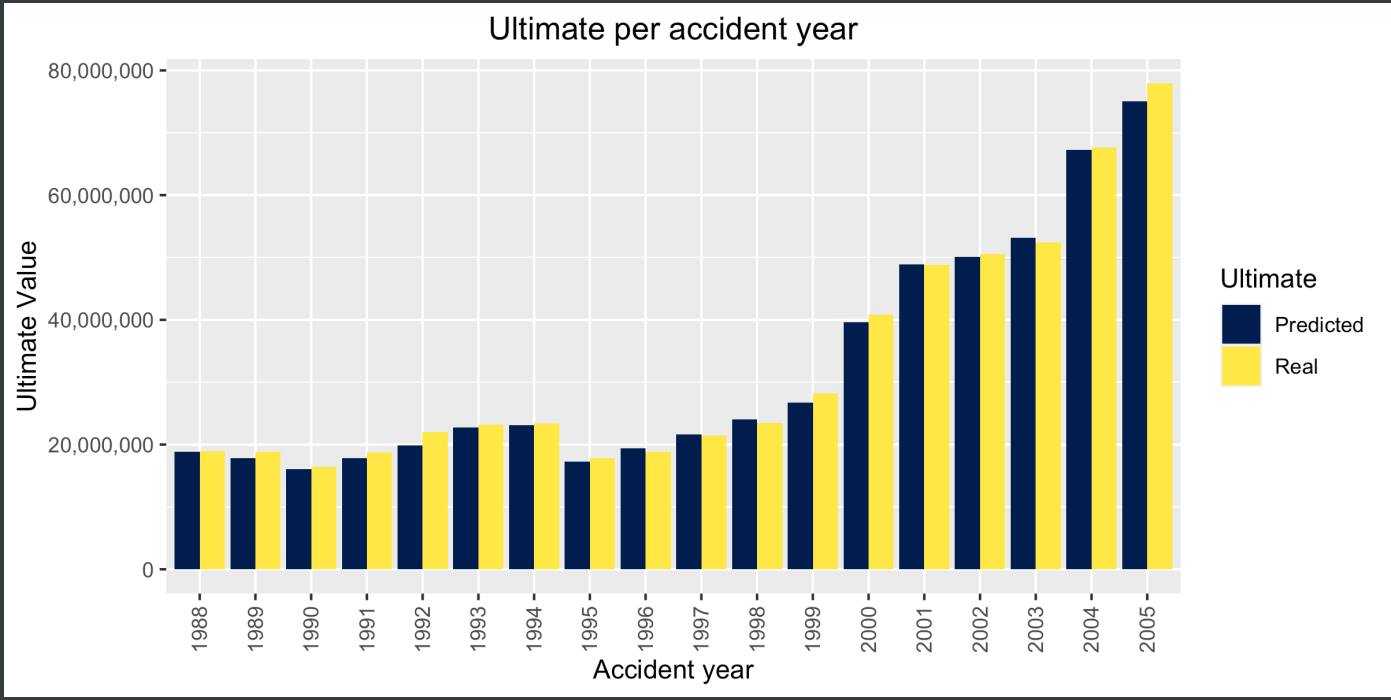
One last remark regarding the methodology we employed is that we found several sets of good parameters. For the final submission we selected the ten best models and averaged their predictions.

Results

The first eight models all scored a root mean square error between 23,350 and 23,450. We do not have access to the true values of the target variable for the test set, but to put this into perspective, in the training set the median ultimate value was roughly 3,300, and the mean roughly 11,000, above the 80% quantile. This is a common signal for few very large claims skewing the whole ultimate distribution.

Analysing the predictions by backtesting on the training set where the model was crossvalidated (RMSE ca. 22,500) suggests that it was largely the presence of these large claims conditioning the RMSE. The distribution of the predictions of the ultimates is much more regular than the distribution of the real ultimates, with a shorter tail and mean closer to the median: small claims tend to have larger estimates than they should, large claims smaller ones. While we recognised this issue already in the modelling phase, we never found an effective way to solve it, the main reason being that for the large claims we looked at in the training set a logic or reason for the exceptionally high amounts was not apparent.

On an aggregated view the model results more reliable and was able to capture market cycles and the inflation tendencies over the years. One can also think of the good result here as the errors of overestimating large claims and underestimating small claims partially cancelling out. Here we compared the real and predicted ultimates on the training set:



Conclusion and future work

In conclusion, the article presents an example of how machine learning can be implemented in the insurance industry, specifically in the reserving process, and discusses some features of the model developed in the ALP competition, and contains some insights on how machine learning in reserving might look like in the future.

All in all we found the results of the model promising - however, there are still areas for improvement and further research that can be done. For example, it would be interesting to explore in more detail other machine learning algorithms and techniques and to see how they compare to XGBoost. More fundamentally, more work should be done to increase the interpretability of the model and to understand how it is making its predictions: prudency and reliability of reserving models are important per se - an imprecise model is often preferred to a stronger one if it is more transparent. If we aim to see more machine learning techniques applied in an actuary's everyday life, the first step must be to make them more accessible and explainable, an issue whose undertones are both mathematical and social.

Overall, I believe that this example serves as an inspiration for further developments in the insurance industry and hope that it will spark discussions among actuaries and data scientists. Machine learning has the potential to revolutionize the way we do reserving and I am excited to see what the future holds in this field.

I would like to thank Attila Gulyas for developing this model with me, the organizers of the ALP for providing us the chance to experience something new, and the whole open source community for planting a long time ago the boosted trees that we were able to employ.

References

The Kaggle page for the ALP can be found at https://www.kaggle.com/c/actuarial-loss-estimation, where the datasets are still available for download. For more information about XGBoost, consult the comprehensive documentation at https://xgboost.readthedocs.io/. Information on spaCy, which we extensively used to analyse the free text claim descriptions, is available on the official website https://spacy.io.