Insurance Claim Size Prediction

Emil Hofman, Jakob Sand Houe, Nick Kjær Mikkelsen & Troels Sabroe Ebbesen

All group members contributed evenly

Overview

- Introduction
- Data
- Models
    - Tree based regression
    - Neural Network regression
- Concluding Remarks and Outlook
- Appendix

Introduction

Context

- To calculate a **risk premium** insurance companies will usually model claim frequency and size/severity separately based on previous claims experience
- In practice, both frequency and severity modelling is done using Generalized Linear Models (GLM):

$$RiskPremium_i = E(F|\mathbf{X}_i)E(S|\mathbf{X}_i) = g(\beta_f^T\mathbf{X}_i)h(\beta_s^T\mathbf{X}_i)$$

for *g*,*h: R -> R* and where *F* and *S* are assumed independent and to follow distributions from the exponential families

Scope

- In this project we only model the claim severity, i.e. $E(S|X)$
- **The goal is to come up with *good* ML alternatives to the run-of-the-mill GLM approach described above**

Data

Comments

- Approximately 30,000 theft/burglary-coverage claims from Codan Forsikring
- Response variable is the total claim cost after accounting for deductibles (‘selvrisiko’)
- Features, in the form of policyholder information, includes:

Categorical Features	
Code	**Description**
Bareal	Home/building size (intervals of m2)
byg_anvend_kode	Home/building category (apartment, house etc.)
geo	Geographical area type
Segment	Policyholder segment
zone	Risk zone (grouped by postal code)
aldersgruppe	Age (intervals)

Continuous Features	
Code	**Description**
afst_politi	Distance to nearest police station
forsum	Sum insured
selvrisk	Deductible

Data

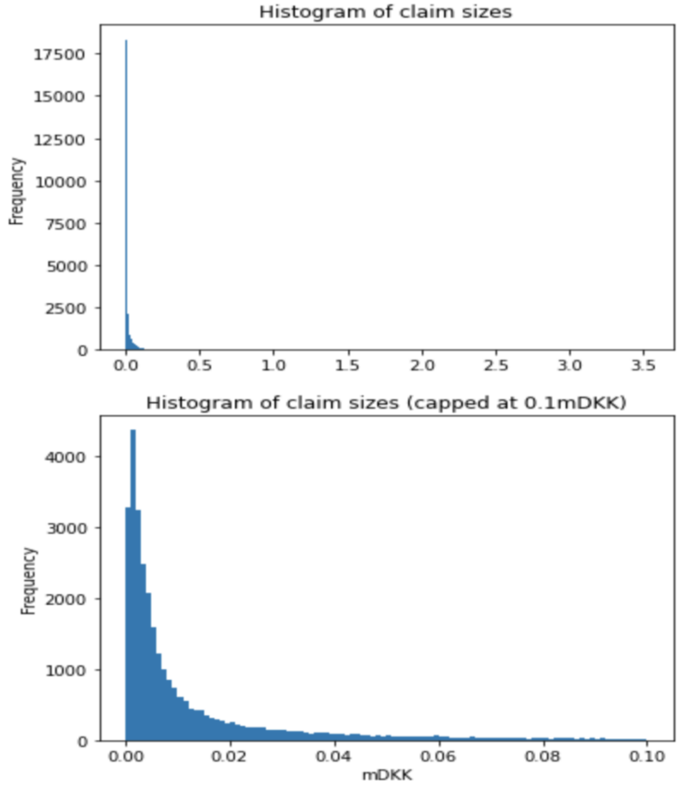
Issues

1. Few very large claims
2. Some factor levels were represented by only a couple of data points (in particular home/building category)
3. Large variance and different order of magnitude of numerical features

Solutions

1. Cap claim sizes at 100,000 DKK
2. Remove commercial-type buildings
3. Normalize and log-transform numerical features where necessary

Claim Sizes

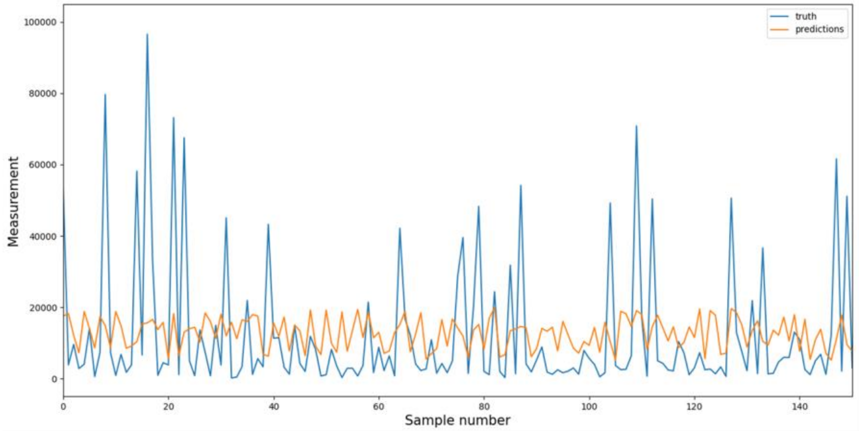


Tree

- Model choice
    - LightGBM
- Categorical variables
    - No one-hot encoding needed
- Model objective
    - Gamma

Tree – Training

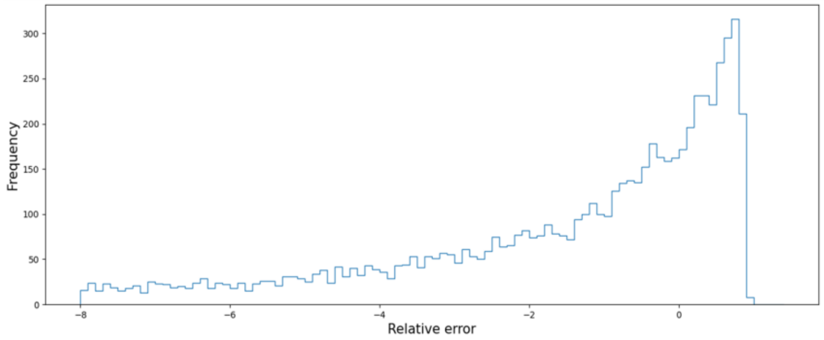
Predictions v Truth



Comments

- Underpredicts large claims
- Over predicts small claims
- Training time : 0.45 sec
- RMSE = 17215

Relative Error

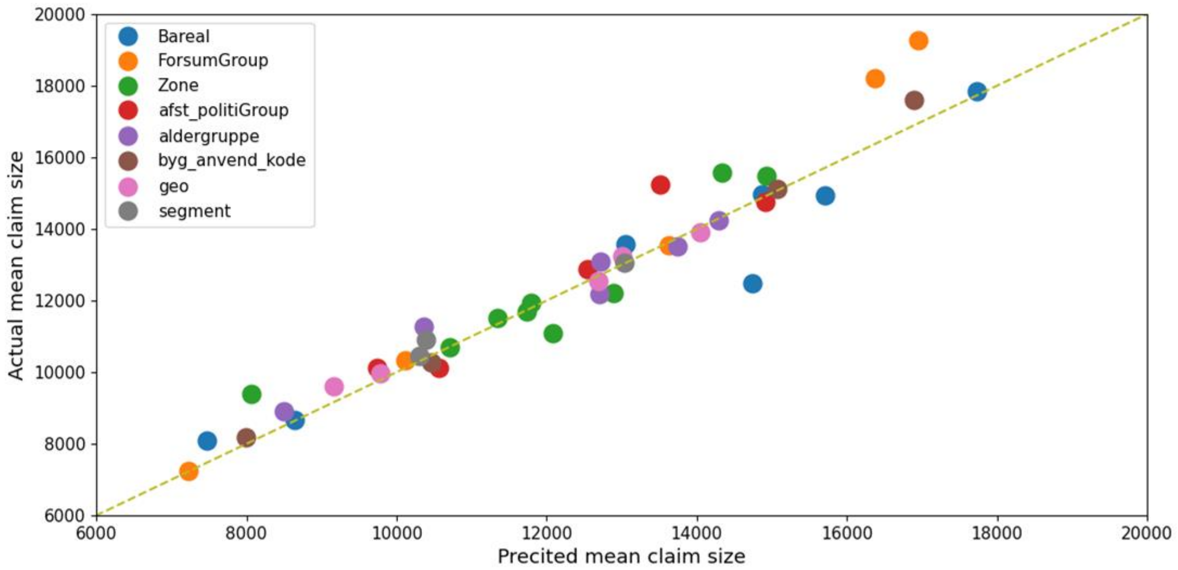


Tree - Grouped

Comments

- Taking the mean and grouping
- Have to keep in mind what kind of product we are dealing with

Predicted Group Means

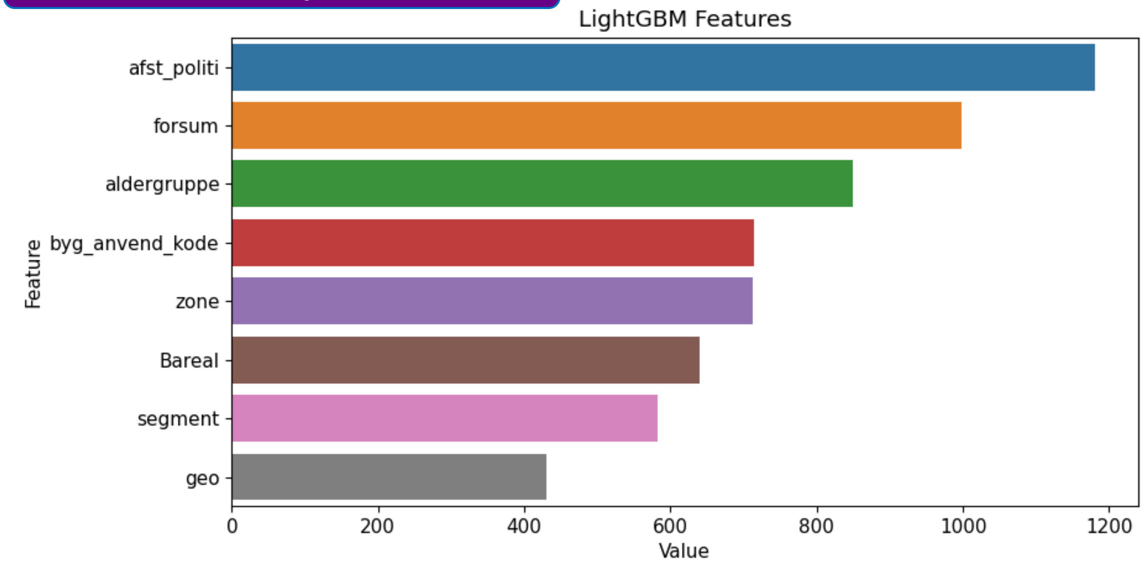


Tree – Feature Importance

Comments

- The ordering is very much what we expected.

Feature Importance



Neural Network

- Numerical variables
  - Log transformed Sum Insured
  - Scaled by subtracting mean and dividing by variance
- Categorical variables
  - One Hot encoded
  - 9 features -> 38 features
- Target variable
  - No transformation

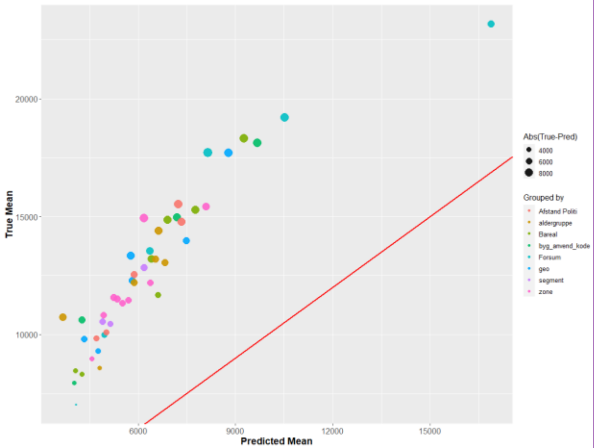
Neural Network – Loss Function

Comments

- Loss Function
  - ‘Outlier’ sensitivity
  - Need the model to *not* ignore larger values
- Red line indicates ‘perfect’ prediction

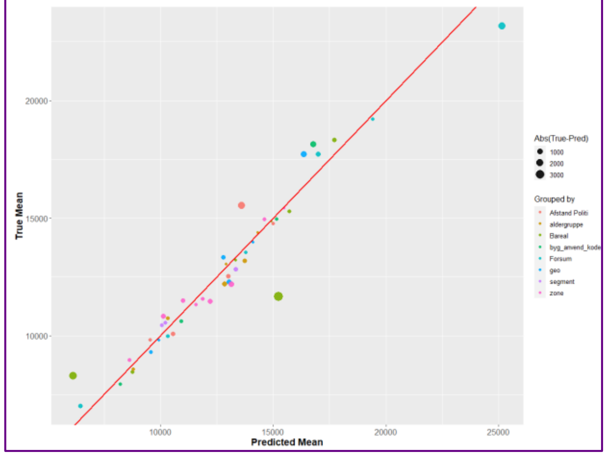
MAE

- Guesses too low



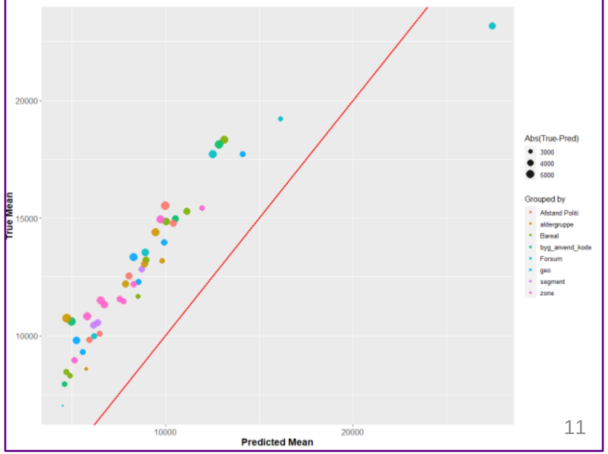
MSE

- Much better fit for the mean



Ber-Hu

- MAE for small loss, close to MSE for larger
- Additional parameter to optimize, ‘when is a loss small?’
- Could not outperform MSE on our data

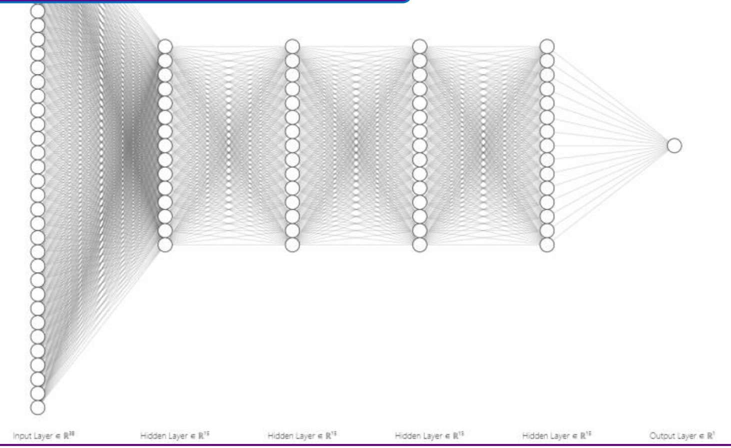


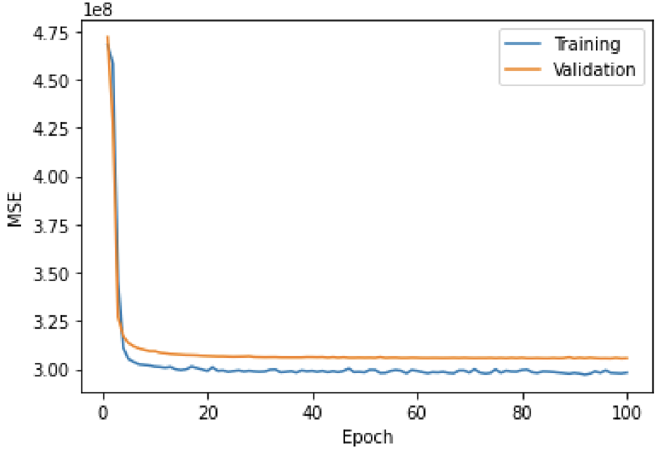
Neural Network – Final Architecture

Comments

- MSE loss function
- Deep vs Shallow
  - Overfit with deep structure, dropout and batch normalisation did not resolve
- Optimised with grid search for final setup (although no CV due to computational cost)
- 15x15x15x1 structure, with ReLU activation and batch normalisation between each layer
- LR 0.001, batch size 256
- Resulting RMSE: Mean 17348 and std.dev. 183 using 5-fold cross validation
- GPU (Google Colab) Training Time: Approx. 53 seconds (pr fold)

Structure and Loss



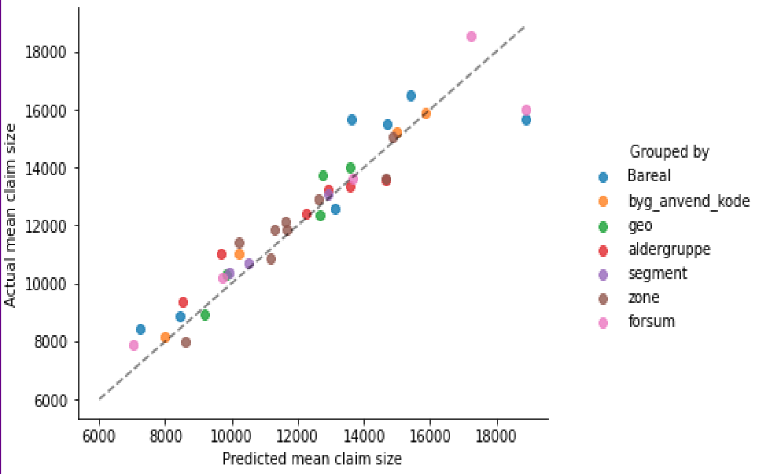


Neural Network – Final Architecture

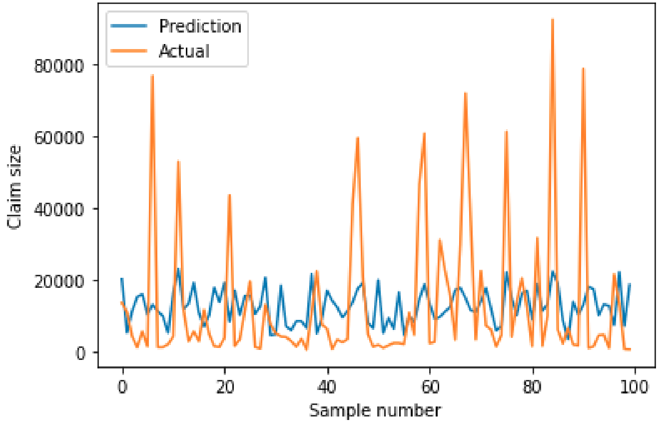
Comments

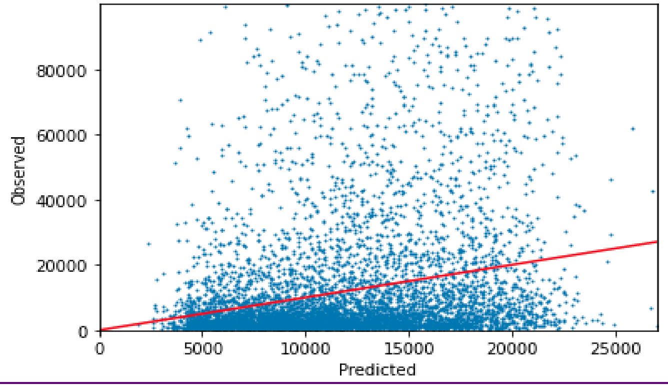
- We are hitting the mean, but neither high nor low values

Predicted Group Means



Predictions





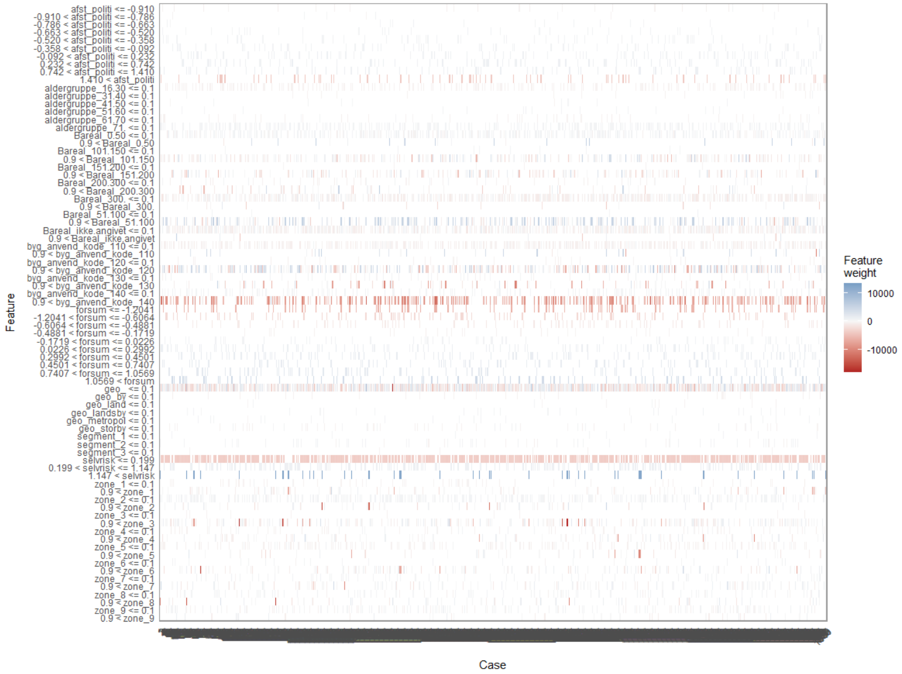
Neural Network – Predictions & Lime

Comments

- LIME, a way to explain machine learning output. Alternative to SHAP and feature importance.

- Most significant feature is a low deductible
- Larger sum insured makes for a larger prediction
- Overall expected result, but deductible perhaps too significant

Lime Output

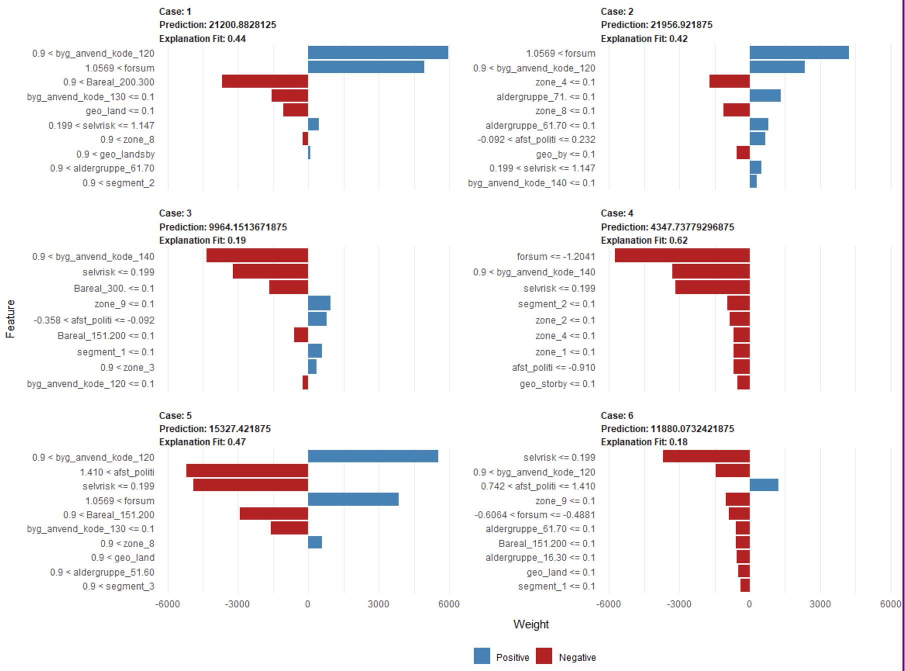


Neural Network – Predictions & Lime

Comments

- 6 Predictions explained
  - 2 high value, 2 low value and 2 around mean
- High predictions dependent on type of house and large sum insured
- Lower prediction lives in apartment and has small deductible.

Lime Output

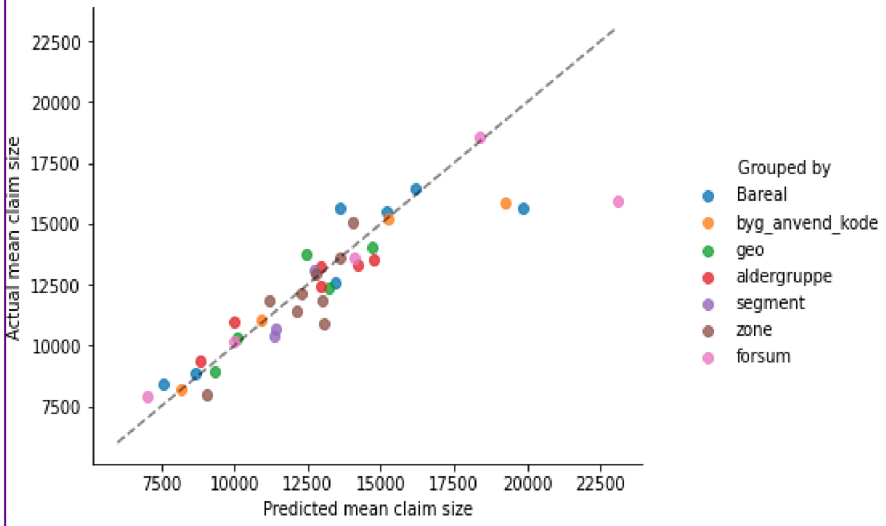


Concluding remarks and outlook

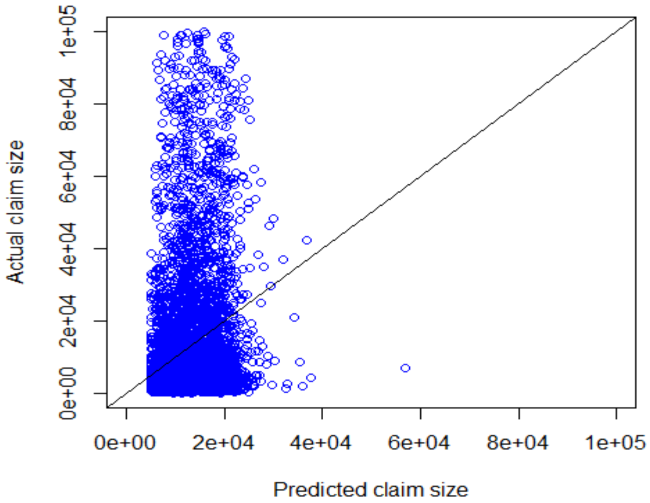
Comments

- Results from a GLM

Predicted Group Means



Claim Size Predictions



Concluding remarks and outlook

Method Comparison

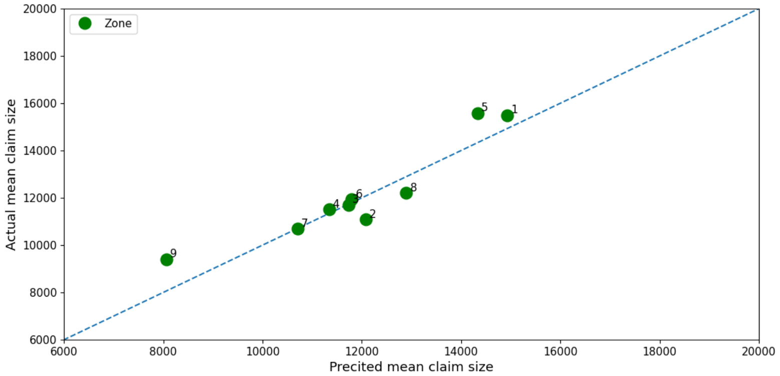
Model	Mean RMSE (5-fold)	Std. Dev. (5-fold)	Computation Time (pr fold)
LGBM	17355	210	0.45 seconds
NN	17348	183	53 seconds
GLM (non-ML)	17354	271	0.44 seconds

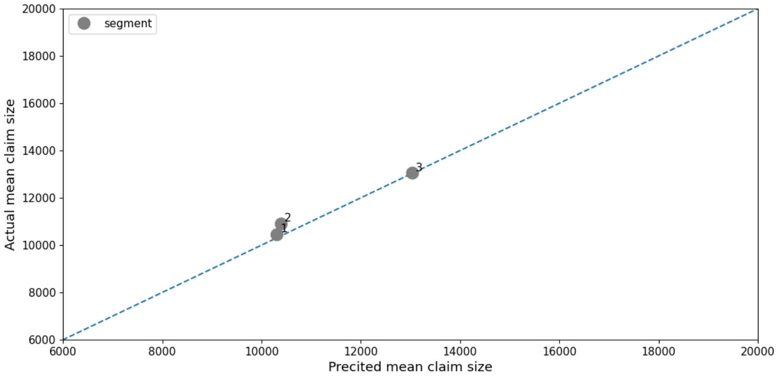
Final Comments

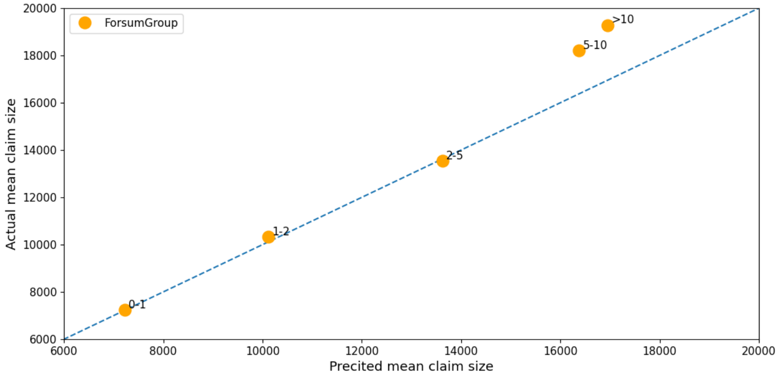
- ML in insurance companies and regulatory issues

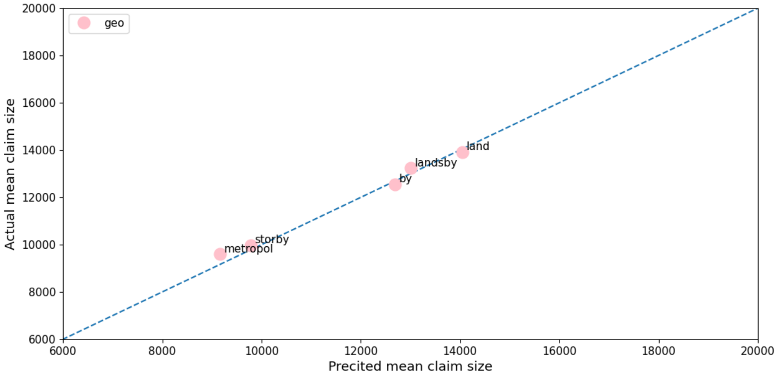
APPENDIX

Tree grouped plots with labels

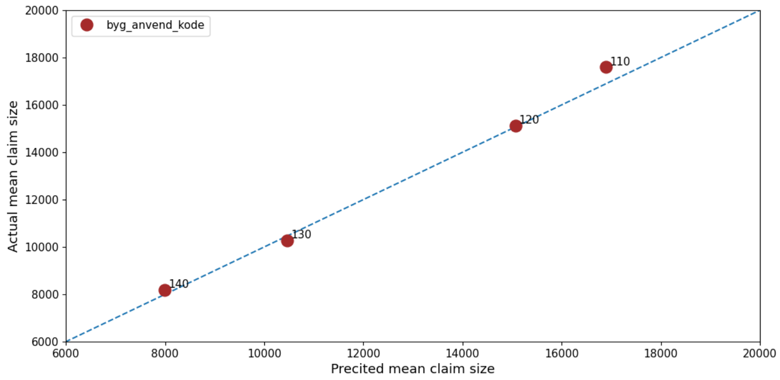


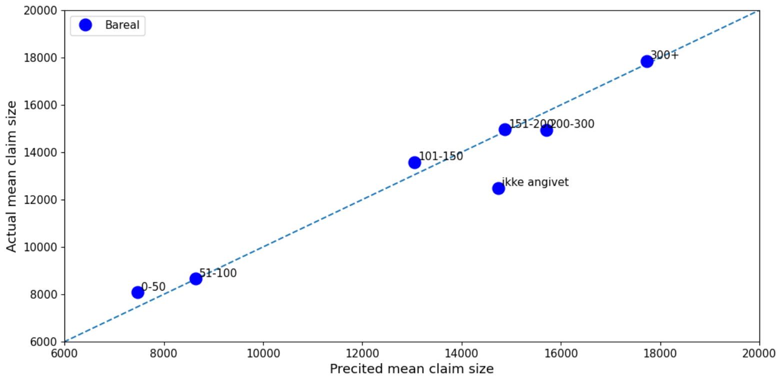


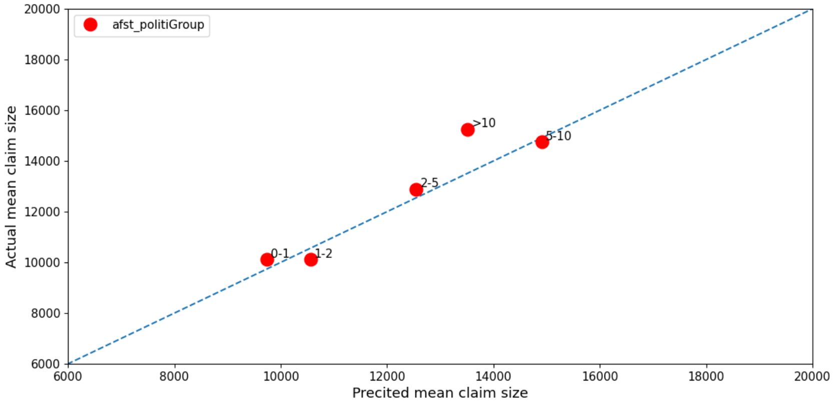


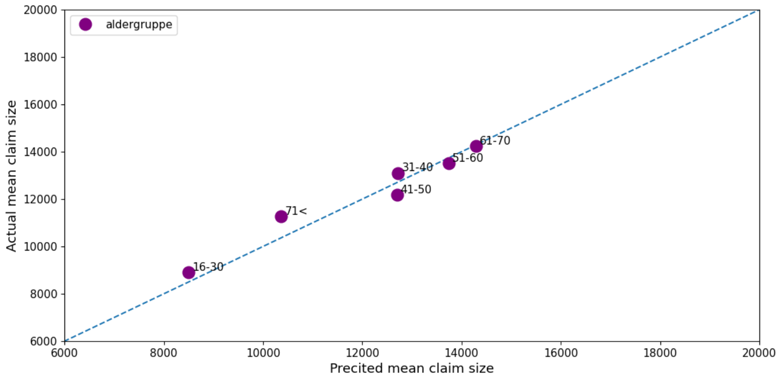


Tree grouped plots with labels









Tree Random search

Ran Random search, for 200 iterations with 5 split cv.

n_estimators was kept at 1000, but early stopping was used in final model.

mean_fit_time	std_fit_time	mean_score_time	std_score_time	bagging_fraction	bagging_freq	feature_fraction	learning_rate	max_depth	min_data_in_leaf	n_estimators	num_leaves	0_test_score	1_test_score	2_test_score	3_test_score	4_test_score	mean_test_score	std_test_score	rank_test_score
0.513	0.044	0.0614	0.0049	60%	4	50%	0.01	6	22	1,000	11	0.0535	0.0583	0.0544	0.0531	0.0565	0.0552	0.0020	1
0.571	0.025	0.0606	0.0012	90%	6	50%	0.01	4	45	1,000	39	0.0526	0.0580	0.0528	0.0540	0.0572	0.0549	0.0022	2
0.455	0.028	0.0473	0.0011	60%	2	50%	0.01	3	48	1,000	192	0.0531	0.0568	0.0536	0.0531	0.0566	0.0546	0.0017	3
0.440	0.029	0.0467	0.0026	60%	8	60%	0.01	3	66	1,000	173	0.0519	0.0575	0.0550	0.0522	0.0557	0.0545	0.0021	4
0.572	0.019	0.0586	0.0010	80%	4	60%	0.01	4	74	1,000	23	0.0511	0.0581	0.0521	0.0535	0.0574	0.0544	0.0028	5
0.478	0.004	0.0451	0.0012	90%	2	60%	0.01	3	17	1,000	186	0.0517	0.0574	0.0541	0.0524	0.0555	0.0542	0.0021	6
0.409	0.015	0.0462	0.0019	60%	8	70%	0.01	3	63	1,000	72	0.0518	0.0582	0.0539	0.0520	0.0551	0.0542	0.0024	7
0.330	0.009	0.0351	0.0004	70%	8	60%	0.02	2	64	1,000	192	0.0520	0.0576	0.0537	0.0524	0.0549	0.0541	0.0020	8
0.607	0.075	0.0608	0.0020	60%	6	70%	0.01	4	45	1,000	57	0.0515	0.0577	0.0528	0.0519	0.0566	0.0541	0.0025	9
0.592	0.042	0.0604	0.0044	80%	4	70%	0.01	4	76	1,000	165	0.0501	0.0581	0.0517	0.0532	0.0574	0.0541	0.0032	10
0.453	0.042	0.0445	0.0010	90%	4	80%	0.01	3	27	1,000	141	0.0508	0.0578	0.0539	0.0529	0.0547	0.0540	0.0023	11
0.423	0.023	0.0393	0.0021	70%	2	50%	0.02	2	69	1,000	46	0.0521	0.0564	0.0537	0.0530	0.0549	0.0540	0.0015	12
0.413	0.018	0.0412	0.0011	60%	6	90%	0.01	3	52	1,000	177	0.0521	0.0578	0.0532	0.0514	0.0544	0.0538	0.0022	13
0.455	0.029	0.0433	0.0026	80%	4	90%	0.01	3	78	1,000	148	0.0502	0.0575	0.0523	0.0532	0.0556	0.0538	0.0025	14
0.317	0.006	0.0341	0.0007	60%	8	80%	0.02	2	64	1,000	15	0.0513	0.0568	0.0535	0.0531	0.0536	0.0536	0.0018	15
0.463	0.040	0.0395	0.0030	90%	6	60%	0.02	2	17	1,000	179	0.0512	0.0574	0.0545	0.0512	0.0539	0.0536	0.0023	16
0.459	0.021	0.0427	0.0004	90%	4	80%	0.01	3	11	1,000	182	0.0504	0.0577	0.0537	0.0516	0.0544	0.0536	0.0025	17
0.719	0.025	0.0826	0.0024	60%	2	50%	0.01	5	20	1,000	156	0.0513	0.0547	0.0529	0.0511	0.0574	0.0535	0.0023	18
0.328	0.015	0.0327	0.0022	70%	6	90%	0.02	2	37	1,000	114	0.0513	0.0574	0.0527	0.0509	0.0546	0.0534	0.0024	19
0.423	0.099	0.0642	0.0387	50%	8	60%	0.01	3	78	1,000	196	0.0512	0.0569	0.0524	0.0517	0.0546	0.0534	0.0021	20
0.387	0.005	0.0352	0.0014	70%	2	70%	0.01	2	42	1,000	183	0.0517	0.0561	0.0541	0.0506	0.0540	0.0533	0.0020	21
0.448	0.098	0.0463	0.0032	50%	8	60%	0.01	3	13	1,000	153	0.0513	0.0572	0.0526	0.0509	0.0544	0.0533	0.0023	22
0.679	0.107	0.0682	0.0090	60%	2	50%	0.02	4	45	1,000	184	0.0496	0.0548	0.0497	0.0525	0.0583	0.0530	0.0033	23
0.291	0.005	0.0331	0.0004	50%	6	80%	0.01	2	60	1,000	46	0.0517	0.0559	0.0535	0.0501	0.0533	0.0529	0.0020	24
0.434	0.011	0.0474	0.0031	80%	6	80%	0.02	3	63	1,000	196	0.0496	0.0568	0.0494	0.0522	0.0565	0.0529	0.0032	25
0.533	0.002	0.0610	0.0010	60%	2	50%	0.02	4	76	1,000	189	0.0495	0.0553	0.0494	0.0521	0.0581	0.0529	0.0034	26

Cross Validation NN

- NN regressor grid search output sorted by loss in ascending order

	Learning rate	Batch size	Number of epochs	Number of layers	Number of neurons per layer	Validation MSE loss	Validation RMSE loss
23	0.0010	256.0	40.0	4.0	15.0	3.087137e+08	17570.250748
47	0.0005	256.0	40.0	4.0	15.0	3.095241e+08	17593.296470
41	0.0005	256.0	30.0	4.0	15.0	3.097660e+08	17600.171798
21	0.0010	256.0	40.0	4.0	5.0	3.100414e+08	17607.993450
18	0.0010	256.0	40.0	3.0	5.0	3.100844e+08	17609.214463
...	...	...	...	...	...	...	...
67	0.0001	256.0	40.0	3.0	10.0	3.753661e+08	19374.365951
61	0.0001	256.0	30.0	3.0	10.0	4.153433e+08	20379.973045
66	0.0001	256.0	40.0	3.0	5.0	4.559609e+08	21353.241971
63	0.0001	256.0	30.0	4.0	5.0	4.646393e+08	21555.494303
60	0.0001	256.0	30.0	3.0	5.0	4.688977e+08	21654.044795

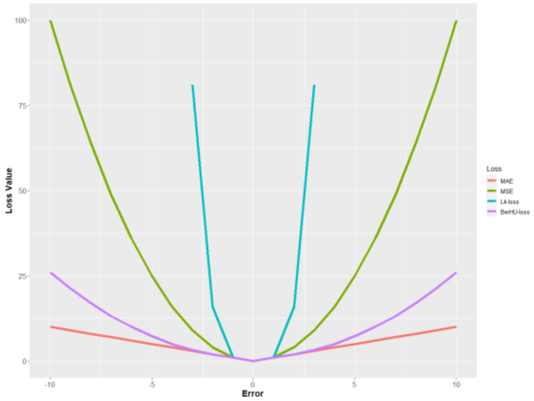
72 rows × 7 columns

Ber-Hu Loss

Based upon : https://arxiv.org/abs/1207.6868

Formula:

$$
\begin{cases} \dfrac{x^2 + c^2}{2c}, & x \geq c \\ |x|, & x < c \end{cases}
$$



Should give more weight to larger errors, but with an adaptive c converge towards MAE.

Need to choose c, in our test we used ⅕ * max(abs(x)) with x being the error.

Lime

Based upon :
https://arxiv.org/abs/1602.04938

Algorithm:

1. Permute observation with slightly different values pr. permutation
2. Compute difference between permutation and true value
3. Predict by selected model on permuted data
4. Select top features to explain prediction
5. Fit simpler regression model based upon selected feature
6. Use resulting feature weights from

For further explanation see https://uc-r.github.io/lime

Selected HP:
n_features = 10,
n_permutations= 5000
dist_fun = “Manhattan”
feature_select = “lasso_path”