Machine learning in general insurance reserving - a worked example

Society of Actuaries in Ireland

Grainne McGuire

10 May 2022

Disclaimer

The views expressed in this presentation are those of the presenter(s) and not necessarily those of the Society of Actuaries in Ireland or their employers.

Machine Learning in Reserving working party

- Who are we?
  - International group of actuaries
- What are our aims?
  - Learn how machine learning (ML) can be used in non-life reserving
  - Carry out research on the use of ML in reserving.
- Various workstreams
  - Foundations
  - Literature Review
  - Survey
  - Data
  - Research
  - Practical Considerations

Find us at https://institute-and-faculty-of-actuaries.github.io/mlr-blog/

MLR

General Insurance Machine learning in Reserving working party

General Insurance Machine Learning in Reserving working party

The General Insurance Machine Learning in Reserving working party is an international group of over 40 actuaries, bringing together experts in this field from around the globe.

Our starting premise is that whilst machine learning techniques are widespread in pricing, they are not being adopted 'on the ground' in reserving (certainly in the UK). The idea of the working party is to help move this forward, by identifying what the barriers are, communicating any benefits, and helping develop the research techniques in pragmatic ways. At the same time we understand the resource and time pressures that reserving actuaries are under and the aim is not to replace existing reserving methods per se, but to start the journey to understanding if and how machine learning may help us in our day to day work.

Our intention is to develop and undertake our own research. To this end, we have a number of

1972 - 2022
Society of Actuaries in Ireland

Foundations Workstream

Provide a path to gaining competency by:

- creating a roadmap of methods to learn
- gathering together relevant learning materials and,
- developing notebooks in R and Python with example code, where the methods are applied to reserving data sets.

Blog articles so far:

- Introduction to R
- My Top 10 R Packages for Data Analysis
- The tidyverse for actuaries
- R's data.table - a useful package for actuaries
- Reserving with GLMs (Python version)
- Self-assembling claim reserving models using the LASSO
- Getting to grips with GLM, GAM and XGBoost

What we are talking about today

Main goal

Illustrate a work flow for:

- setting up a data set for machine learning
- applying different ML models to this data
- tuning the ML hyper-parameters
- comparing and contrasting performance for the past fitted values and the future predictions.

Based on work done with Jacky Poon on behalf of MLRWP

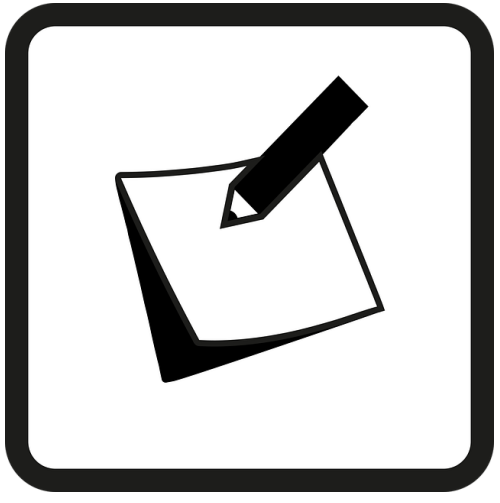
Secondary goal

- Demonstrate the utility of a machine learning framework which enables the user to easily plug and play different machine learning models

Talk outline

1. The data set
2. The mlr3 machine learning framework
3. Data preparation
4. Tuning process
5. Fitting some models
6. Model analysis
7. Conclusions
8. Questions?

- Blog post based on this talk with code in R
- Python version



1 - The Data Set

The data set

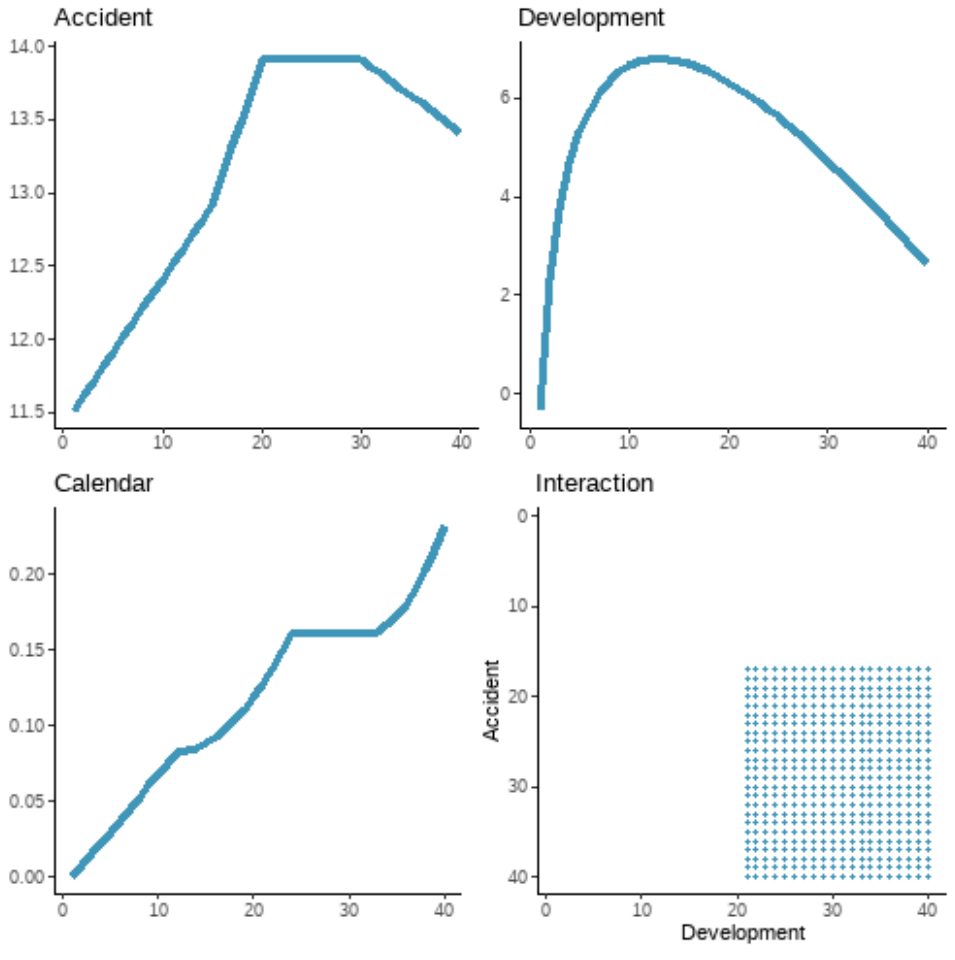
We use simulated data (data set 3 from this paper)

- 40 x 40 triangle of aggregated incremental payments (all positive)
- Features are accident / development / calendar quarters
- Characteristics shown in the plot

Benefits of simulated data

- We know the future
- We can control the future
- Data structure is familiar to us

But a small data set may not showcase fully benefits of ML



What the data look like

The first few rows

pmts	acc	dev	cal	mu	train_ind
242,671	1	1	1	71,653	TRUE
164,001	1	2	2	1,042,776	TRUE
3,224,478	1	3	3	4,362,600	TRUE
3,682,531	1	4	4	10,955,670	TRUE
10,149,369	1	5	5	20,800,545	TRUE
28,578,275	1	6	6	33,089,167	TRUE

and the last few

pmts	acc	dev	cal	mu	train_ind
125,261,750	40	35	74	109,039,367	FALSE
62,657,370	40	36	75	82,853,302	FALSE
63,467,681	40	37	76	62,682,720	FALSE
26,041,979	40	38	77	47,227,843	FALSE
33,947,274	40	39	78	35,444,881	FALSE
37,258,687	40	40	79	26,503,298	FALSE

2 - The mlr3 Machine Learning Framework

The idealised modelling workflow

Task

Dataset to analyse

- Features
- Target variable
- Validation split

Data Sampler

Sampling approach

- Train / Test
- Cross Validate

Learner

Common, consistent framework

- Train
- Predict

Tuner

Hyper parameter search

- Grid search,

etc.

Machine Learning Frameworks

- ML modelling (in R) usually involves using different packages
  - different input specifications for data
  - different output types
- This makes it complex to work with or test multiple models, especially in a pipeline.
- But ML multimodel frameworks allow you to plug and play different models. Examples include:
  - caret (R)
  - tidymodels (R) - a successor to caret
  - mlr3 (R)
  - scikit-learn (Python)
- When picking one, read up on a few and select the one that works the best for you.
- We will use mlr3 - the successor to the popular but no longer actively developed mlr.

How mlr3 operates

mlr3 uses the R6 object orientated classes - R6 is similar to object orientated programming in other languages such as Python or C++, but is quite different to R's S3 and S4 object orientated approaches.

mlr3 objects contain both the data and the methods (functions) used to manipulate them.

- `<object name>$train()` to train
- `<object name>$model` to access the model
- `<object name>$predict()` to predict, etc...
- Syntax looks like `<object name>$<data name>` or `<object name>$<function name>`

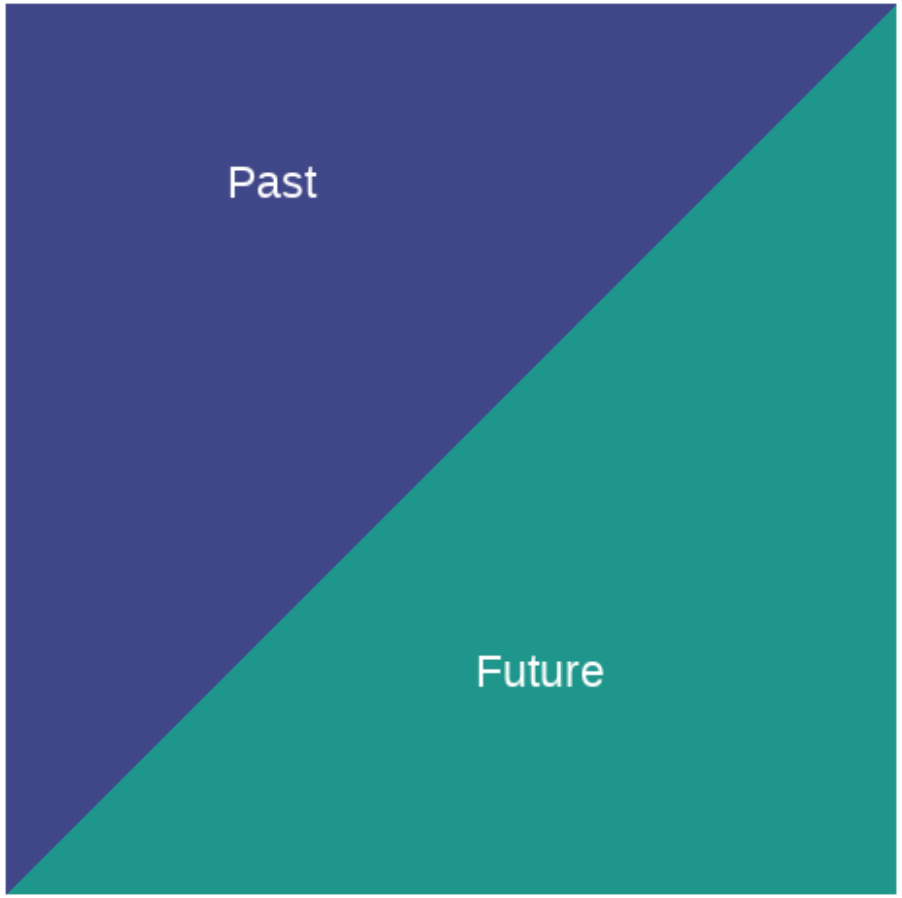
Resources:

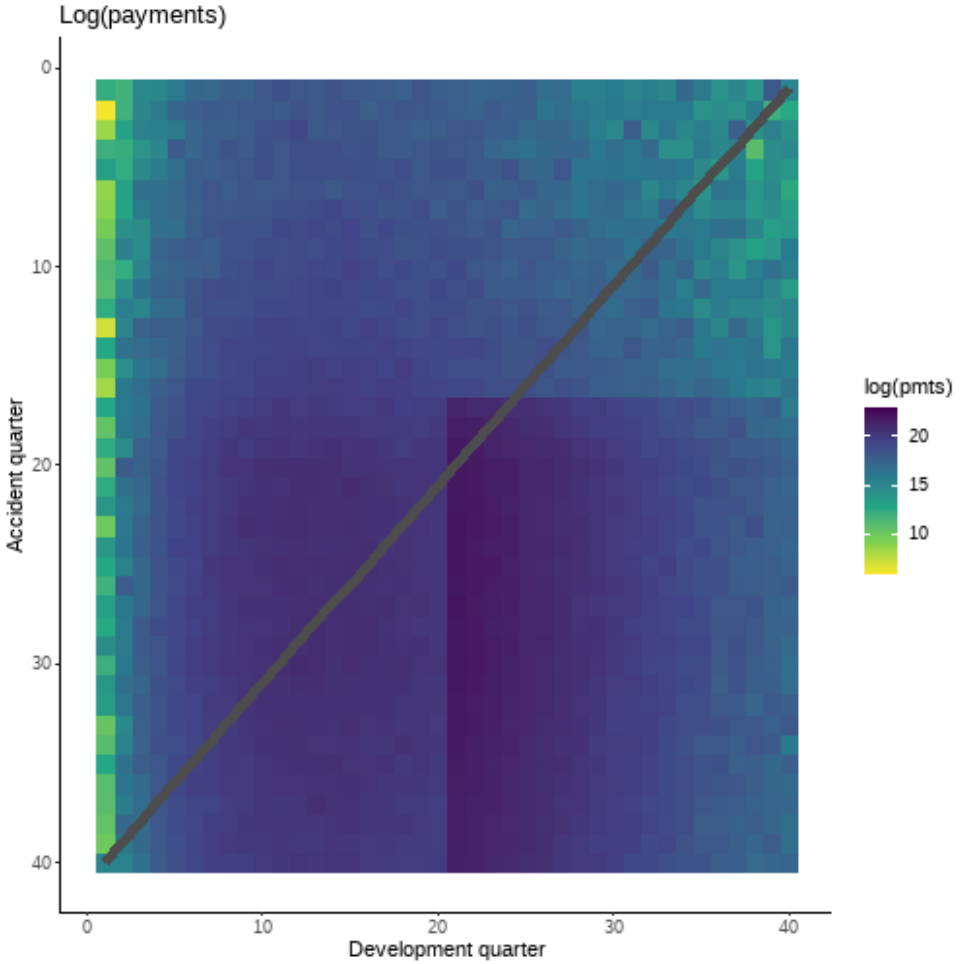
- Get help by using the `help()` method
  - `mlr3_obj$help()` for an **mlr3** object called `mlr3_obj`. E.g. `lrn("regr.rpart")$help()`
- The mlr3 book
- Cheatsheets
- Gallery of examples

3 - Data Preparation

1972 - 2022
Society of Actuaries in Ireland

A look at the data





Create an mlr3 task

```r
#  add row_id
dat[, row_id := 1:.N]

task <- TaskRegr$new(id = "reserves",
                     backend = dat,
                     target = "pmts")

# look at the task
task
```

```
## <TaskRegr:reserves> (1600 x 7)
## * Target: pmts
## * Properties: -
## * Features (6):
##   - int (4): acc, cal, dev, row_id
##   - dbl (1): mu
##   - lgl (1): train_ind
```

But, we only want to use `acc`, `cal` and `dev`.

So, sort out row and column roles

```r
task$set_row_roles(dat[train_ind==FALSE, row_id], roles="valid

# make row_id a name
task$set_col_roles("row_id", roles="name")

# drop train_ind and mu from the feature list
task$col_roles$feature <- setdiff(task$feature_names, c("train

# Check the task has the correct variables now
# You can also check the column roles after with `task$col_rol
task
```

```
## <TaskRegr:reserves> (820 x 4)
## * Target: pmts
## * Properties: -
## * Features (3):
##   - int (3): acc, cal, dev
```

Models fitted in mlr3

With the framework set up, we can try different models. For each model we:

1. select hyper-parameters for tuning
2. define the search space over the hyper-parameters
3. evaluate using cross-validation
4. select the hyper-parameters that produce the lowest RMSE for the cross-validation
5. create a final model fitting to all the data and use this to assess performance on the future data (the lower triangle)

The models we will try in **mlr3** are:

- Decision tree
- Random forest
- XGBoost

Also compare results to:

- Chain ladder (fitted via GLM)
- LASSO

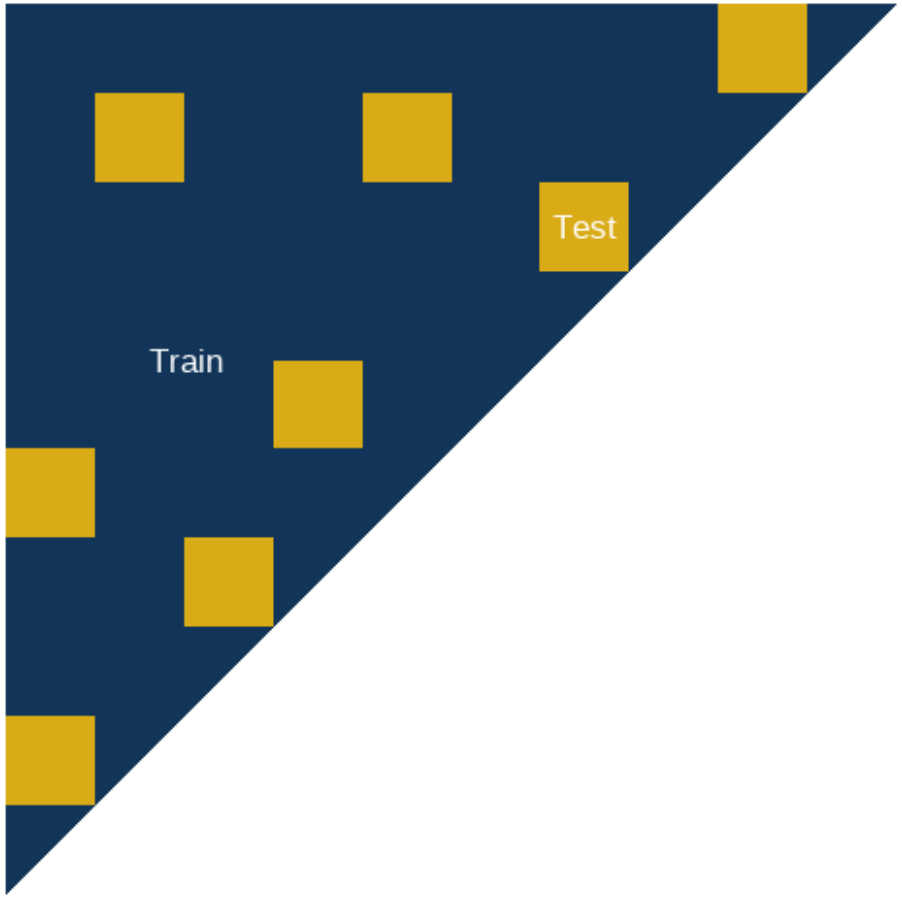
4 - Tuning Process

Background

- Machine learning models have parameters and hyper-parameters
- ML fits the parameters automatically.
- The hyper-parameters control this.
- Tuning these can lead to different quality of fits
- Use train and test data to prevent over-fitting.

Chain-ladder as an ML structure

- The age-to-age factors are the parameters
- The averaging period is the hyper-parameter - once we set this, the algorithm knows how to fit the model
- Varying the averaging period will impact the model fit



What are we validating?

We need to define:

- a set of parameters to tune
- a range of values to consider for these

The paradox library allows us to do this.

We need to have a performance measure.

- `mlr_measures$print()` shows all options
- `?mlr_measures_regr.rmse` for specific description

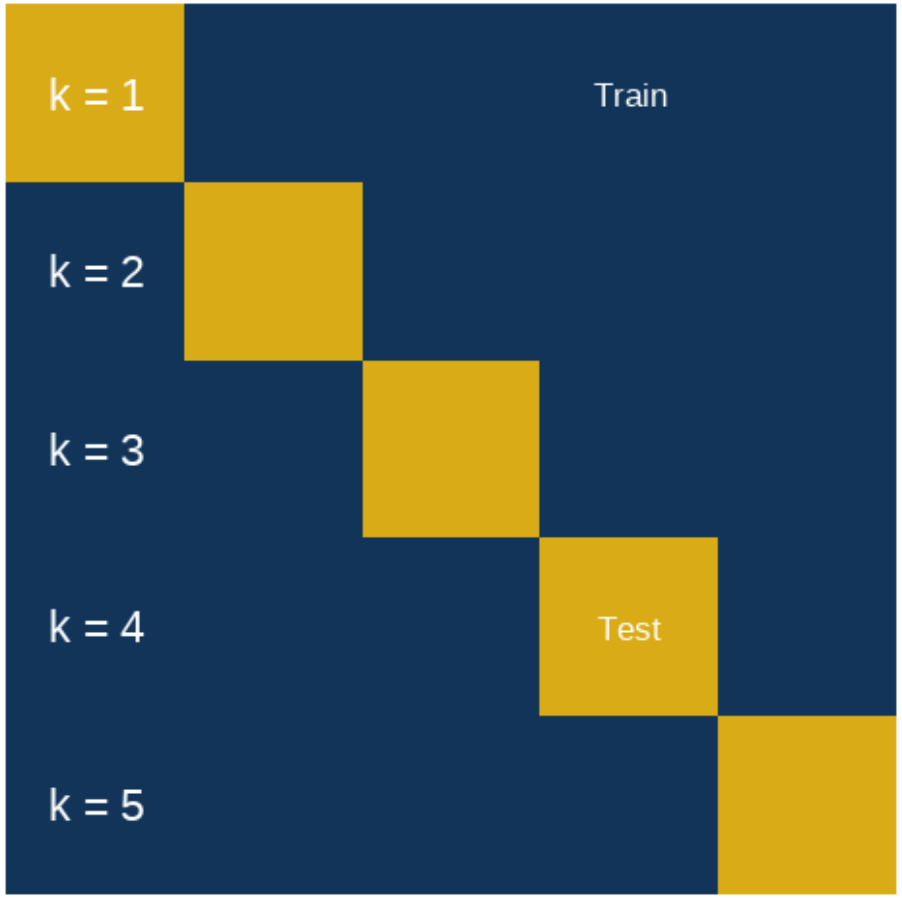
```
# List of all available measures
mlr_measures$print()
```

```
## <DictionaryMeasure> with 54 stored values
## Keys: classif.acc, classif.auc, classif.bacc, classif.bbrier,
##   classif.ce, classif.costs, classif.dor, classif.fbeta, classif.fdr
##   classif.fn, classif.fnr, classif.fomr, classif.fp, classif.fpr,
##   classif.logloss, classif.mbrier, classif.mcc, classif.npv,
##   classif.ppv, classif.prauc, classif.precision, classif.recall,
##   classif.sensitivity, classif.specificity, classif.tn, classif.tnr,
##   classif.tp, classif.tpr, debug, oob_error, regr.bias, regr.ktau,
##   regr.mae, regr.mape, regr.maxae, regr.medae, regr.medse, regr.mse,
##   regr.msle, regr.pbias, regr.rae, regr.rmse, regr.rmsle, regr.rrse,
##   regr.rse, regr.rsq, regr.sae, regr.smape, regr.srho, regr.sse,
##   selected_features, time_both, time_predict, time_train
```

We will use RMSE.

```
measure <- msr("regr.rmse")
```

Cross-validation



Cross-validation (CV) in **mlr3**

```
# create object
crossval = rsmp("cv", folds=6)

set.seed(42)  # reproducible results

# populate the folds
crossval$instantiate(task)
```

For those unfamiliar with CV, it is a sampling approach to estimate performance "out-of-sample":

- Split data into k pieces
- Pick 1 piece to measure on, train model on remaining $k-1$ pieces
- Repeat k times to get measure on all k pieces

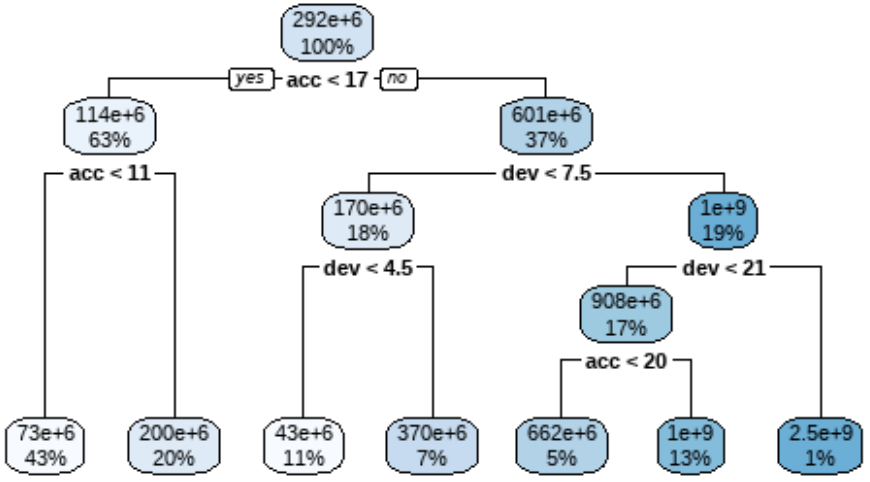
5 - Fitting Models

1972 - 2022
Society of Actuaries in Ireland

Decision Tree

Fit a default tree and visualise

```
lrn_rpart_default <- lrn("regr.rpart")
lrn_rpart_default$train(task)
rpart.plot::rpart.plot(lrn_rpart_default$model,
                       roundint = FALSE)
```



Select parameters for tuning

```
lrn_rpart_default$param_set %>%
  as.data.table() %>%
  kable()
```

id	class	lower	upper	levels	nlevels	is_bounded	special_vals	default	storage_type	tags
minsplit	ParamInt	1	Inf	NULL	Inf	FALSE	NULL	20	integer	train
minbucket	ParamInt	1	Inf	NULL	Inf	FALSE	NULL	<environment: 0x0000000015faa3a8>	integer	train
cp	ParamDbl	0	1	NULL	Inf	TRUE	NULL	0.01	numeric	train
maxcompete	ParamInt	0	Inf	NULL	Inf	FALSE	NULL	4	integer	train
maxsurrogate	ParamInt	0	Inf	NULL	Inf	FALSE	NULL	5	integer	train
maxdepth	ParamInt	1	30	NULL	30	TRUE	NULL	30	integer	train
usesurrogate	ParamInt	0	2	NULL	3	TRUE	NULL	2	integer	train
surrogatestyle	ParamInt	0	1	NULL	2	TRUE	NULL	0	integer	train
xval	ParamInt	0	Inf	NULL	Inf	FALSE	NULL	10	integer	train
keep_model	ParamLgl	NA	NA	TRUE, FALSE	2	TRUE	NULL	FALSE	logical	train

Tuning the decision tree

Set the parameter search space - grid of 5 x 5, tuning: `cp` (complexity parameter) and `minsplit` (min obs to split):

```
tune_ps_rpart <- ps(
  cp = p_dbl(lower = 0.001, upper = 0.1),
  minsplit = p_int(lower = 1, upper = 10)
)

# to see what's searched if we set a grid of 5
rbindlist(generate_design_grid(tune_ps_rpart, 5)$transpose())
```

```
##          cp minsplit
##  1: 0.00100        1
##  2: 0.00100        3
##  3: 0.00100        5
##  4: 0.00100        8
##  5: 0.00100       10
##  6: 0.02575        1
##  7: 0.02575        3
##  8: 0.02575        5
##  9: 0.02575        8
## 10: 0.02575       10
## 11: 0.05050        1
## 12: 0.05050        3
## 13: 0.05050        5
## 14: 0.05050        8
## 15: 0.05050       10
## 16: 0.07525        1
## 17: 0.07525        3
## 18: 0.07525        5
## 19: 0.07525        8
```

Set up the tuning

```
evals_trm = trm("evals", n_evals = 25) # Run 25 sets only

instance_rpart <- TuningInstanceSingleCrit$new(
  task = task,
  learner = lrn("regr.rpart"),
  resampling = crossval,
  measure = measure,
  search_space = tune_ps_rpart,
  terminator = evals_trm
)

tuner <- tnr("grid_search", resolution = 5)
```

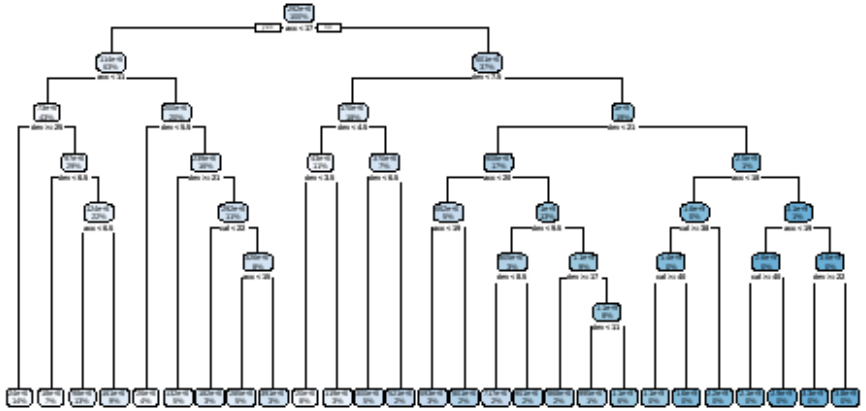
Run the tuning

```
tuner$optimize(instance_rpart)
```

Fit the final model:

```
lrn_rpart_tuned <- lrn("regr.rpart")
lrn_rpart_tuned$param_set$values =
  instance_rpart$result_learner_param_vals
lrn_rpart_tuned$train(task)
```

```
rpart.plot::rpart.plot(lrn_rpart_tuned$model,
                       roundint = FALSE)
```



Random forest - ranger

The model has a number of parameters (which won't all fit on the slide), but we will only tune the parameters `num.trees` and `min.node.size`.

`max.depth`, `mtry` and `sample.fraction` are also often helpful to tune.

```
lrn("regr.ranger")$param_set %>% as.data.table() %>% kable()
```

id	class	lower	upper	levels	nlevels	is_bounded	special_vals	default	storage_type	tags
num.trees	ParamInt	1	Inf	NULL	Inf	FALSE	NULL	500	integer	train , predict
mtry	ParamInt	1	Inf	NULL	Inf	FALSE	NULL	<environment: 0x00000000162ed840>	integer	train
importance	ParamFct	NA	NA	none , impurity , impurity_corrected, permutation	4	TRUE	NULL	<environment: 0x0000000016272760>	character	train
write.forest	ParamLgl	NA	NA	TRUE, FALSE	2	TRUE	NULL	TRUE	logical	train
min.node.size	ParamInt	1	Inf	NULL	Inf	FALSE	NULL	5	integer	train
replace	ParamLgl	NA	NA	TRUE, FALSE	2	TRUE	NULL	TRUE	logical	train
sample.fraction	ParamDbl	0	1	NULL	Inf	TRUE	NULL	<environment: 0x00000000160b5a38>	numeric	train
splitrule	ParamFct	NA	NA	variance , extratrees, maxstat	3	TRUE	NULL	variance	character	train
num.random.splits	ParamInt	1	Inf	NULL	Inf	FALSE	NULL	1	integer	train
alpha	ParamDbl	-Inf	Inf	NULL	Inf	FALSE	NULL	0.5	numeric	train
minprop	ParamDbl	-Inf	Inf	NULL	Inf	FALSE	NULL	0.1	numeric	train
split.select.weights	ParamDbl	0	1	NULL	Inf	TRUE	NULL	<environment: 0x0000000015e65ae0>	numeric	train
always.split.variables	ParamUty	NA	NA	NULL	Inf	FALSE	NULL	<environment: 0x0000000015ddb238>	list	train
respect.unordered.factors	ParamFct	NA	NA	ignore , order , partition	3	TRUE	NULL	ignore	character	train
keep.inbag	ParamLgl	NA	NA	TRUE, FALSE	2	TRUE	NULL	FALSE	logical	train
holdout	ParamLgl	NA	NA	TRUE, FALSE	2	TRUE	NULL	FALSE	logical	train
num.threads	ParamInt	1	Inf	NULL	Inf	FALSE	NULL	<environment: 0x0000000015c26410>	integer	train , predict

Random forest - fit final model

Tune ranger:

```
tune_ps_ranger <- ps(
  num.trees = p_int(lower = 100, upper = 900),
  min.node.size = p_int(lower = 1, upper = 5)
)
instance_ranger <- TuningInstanceSingleCrit$new(
  task = task,
  learner = lrn("regr.ranger"),
  resampling = crossval,
  measure = measure,
  search_space = tune_ps_ranger,
  terminator = evals_trm
)
tuner$optimize(instance_ranger)
```

Fit the final model:

```
lrn_ranger_tuned <- lrn("regr.ranger")
lrn_ranger_tuned$param_set$values = instance_ranger$result_lea

lrn_ranger_tuned$train(task)
```

The model:

```
lrn_ranger_tuned$model
```

```
## Ranger result
##
## Call:
##  ranger::ranger(dependent.variable.name = task$target_names, data =
##
## Type:                             Regression
## Number of trees:                  300
## Sample size:                      820
## Number of independent variables:  3
## Mtry:                             1
## Target node size:                 2
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       14255125223769820
## R squared (OOB):                  0.9224571
```

XGBoost

```
tune_ps_xgboost <- ps(
  objective = p_fct("reg:tweedie"),
  tweedie_variance_power = p_dbl(lower = 1.01, upper = 1.99),
  eta = p_dbl(lower = 0.01, upper = 0.3),
  gamma = p_dbl(lower = 0, upper = 0),
  max_depth = p_int(lower = 2, upper = 6),
  nrounds = p_int(lower = 100, upper = 500)
)

instance_xgboost <- TuningInstanceSingleCrit$new(
  task = task,
  learner = lrn("regr.xgboost"),
  resampling = crossval,
  measure = measure,
  search_space = tune_ps_xgboost,
  terminator = evals_trm
)
```

- Tuning more parameters (4)
- But $4^5 = 1024$ grid - that will take a while to run...
- Quicker search strategy - use random search

```
set.seed(84)  # for random search for reproducibility
tuner <- tnr("random_search")
tuner$optimize(instance_xgboost)
```

Fitting XGBoost

Best fits we have found so far (when running a lot more evaluations:

Hyperparameter	Best	Alternative
nrounds	233	265
tweedie_variance_power	1.01	1.011768
eta	0.3	0.2310797
gamma[fixed]	0	0
max_depth	3	3

Fit both models (right):

```
# Best model from this code:
lrn_xgboost_tuned = lrn(
  "regr.xgboost",
  objective="reg:tweedie",
  nrounds=265,
  tweedie_variance_power=1.011768,
  eta=0.2310797,
  gamma=0,
  max_depth=3)
lrn_xgboost_tuned$train(task)
```

```
# And the version with the parameter set we identified
# before with a larger random search:
lrn_xgboost_prevtuned = lrn(
  "regr.xgboost",
  objective="reg:tweedie",
  nrounds=233,
  tweedie_variance_power=1.01,
  eta=0.3,
  gamma=0,
  max_depth=3)
lrn_xgboost_prevtuned$train(task)
```

Other models

Also compare results to:

- the Chainladder (fitted via GLM)
- a LASSO model (developed with Greg Taylor and Hugh Miller)

We could fit these models using **mlr3** but different features are needed.

Chainladder reserve

An `acc+dev` over-dispersed Poisson GLM fitted to incremental payments gives the same reserve estimates as a volume-all Chainladder.

```
dat[, accf := as.factor(acc)
    ][, devf := as.factor(dev)]

cl <- glm(data=dat[train_ind==TRUE, .(pmts, accf, devf)],
          formula=pmts ~ accf + devf,
          family=quasipoisson(link="log"))

summary(cl)
```

```
##
## Call:
## glm(formula = pmts ~ accf + devf, family = quasipoisson(link = "log"
##     data = dat[train_ind == TRUE, .(pmts, accf, devf)])
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -24314.2   -2132.8    -165.5    2197.1   29768.4
##
## Coefficients:
##             Estimate Std. Error t value             Pr(>|t|)
## (Intercept)  10.4872     1.5323   6.844  0.00000000001614275 ***
## accf2         0.1081     0.1850   0.584             0.559170
## accf3         0.2943     0.1790   1.644             0.100514
## accf4         0.3142     0.1783   1.762             0.078428 .
## accf5         0.4975     0.1722   2.889             0.003979 **
## accf6         0.3966     0.1757   2.257             0.024300 *
## accf7         0.5812     0.1697   3.425             0.000648 ***
## accf8         0.7639     0.1646   4.641  0.00000409939508379 ***
## accf9         0.8160     0.1634   4.993  0.00000074032195234 ***
## accf10        0.9134     0.1612   5.666  0.00000002087163522 ***
## accf11        1.0424     0.1585   6.577  0.00000000009061906 ***
## accf12        1.0535     0.1584   6.649  0.00000000005719225 ***
## accf13        1.2365     0.1550   7.976  0.00000000000000573 ***
## accf14        1.3824     0.1527   9.052 < 0.0000000000000002 ***
## accf15        1.4433     0.1521   9.491 < 0.0000000000000002 ***
## accf16        1.6146     0.1499  10.772 < 0.0000000000000002 ***
## accf17        2.3237     0.1430  16.246 < 0.0000000000000002 ***
## accf18        2.5606     0.1418  18.061 < 0.0000000000000002 ***
## accf19        2.6943     0.1415  19.045 < 0.0000000000000002 ***
## accf20        2.7579     0.1418  19.446 < 0.0000000000000002 ***
## accf21        2.6739     0.1436  18.627 < 0.0000000000000002 ***
```

Overview of the LASSO

GLMs

The loss function associated with a GLM regression is the **deviance**:

$$D(y; X, \hat{\beta}) = -2 \sum_{i=1}^{N} \ell(y_i; X, \hat{\beta}) + \text{other terms}$$

where:

- y is the *N*-vector of observations y_i
- X is the regression design matrix
- β is the *p*-vector of coefficients β_j in the linear predictor
- $\hat{\beta}$ is the regression estimate of β
- $\ell(y_i; X, \hat{\beta})$ is the log-likelihood of observation y_i

LASSO

The loss function for the LASSO version of this GLM is:

$$\mathcal{L}(y; X, \hat{\beta}) = D(y; X, \hat{\beta}) + \lambda ||\hat{\beta}||_1$$

where:

- $\lambda ||\hat{\beta}||_1$ is the parameter penalty or regularisation
- $||\hat{\beta}||_1 = \sum_{j=1}^{p} |\hat{\beta}_j|$ is the L_1 norm of $\hat{\beta}$
- $\lambda \geq 0$ is a tuning parameter controlling the size of the penalty and therefore the amount of the regularisation.

Self-assembling loss reserve models

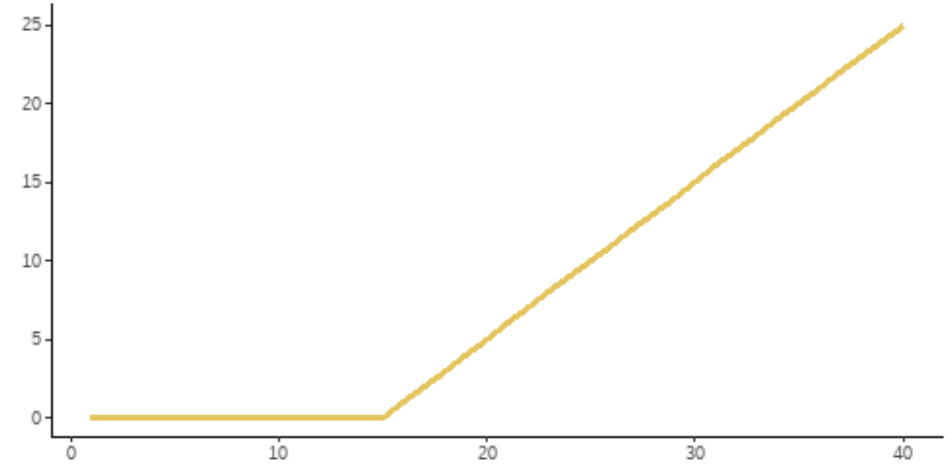
- Described here
- Automated way to create a GLM-like model
- Feature engineering of acc/dev/cal parameters into building block basis functions
  - ramp functions (main effects)
  - step functions (interactions)
- 3 x 39 ramp functions, e.g.

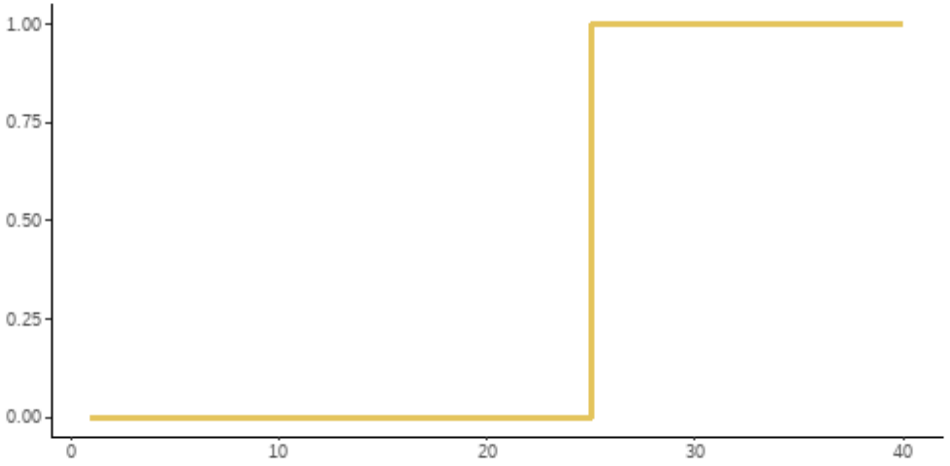
$$\max(\text{acc} - k, 0)$$

- 3 x 39 x 39 interactions

$$I(\text{acc} \geq k) * I(\text{dev} \geq j)$$

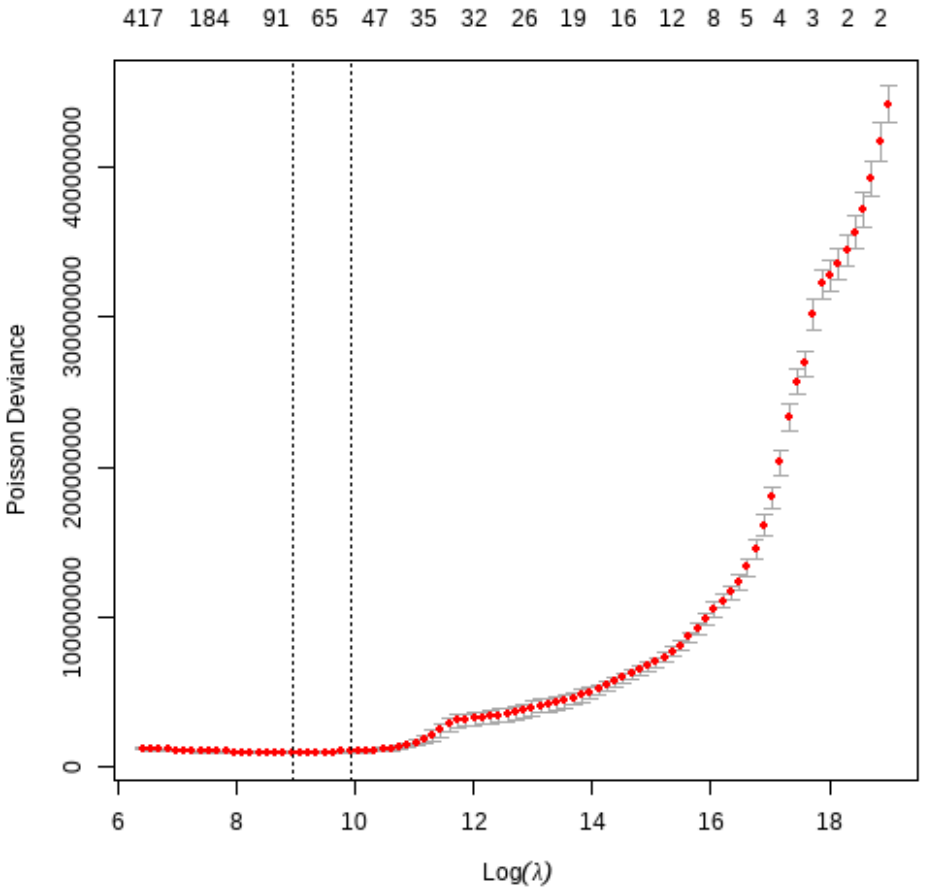
- Similar for (acc, cal) and (dev, cal)
- These are then scaled





Fit the lasso

- Blog post provides all details on
  - how to create the basis functions
  - how to fit the model (using mlr3 and using the underlying package glmnet directly)
- Fit done using 6-fold cross-validation
  - CV done within glmnet so folds different to those used for other models
  - Performance measure is also different - Poisson deviance
- Select model with lowest CV error



6 - Model Analysis

RMSE results

- Calculate all predictions (past and future) for all models
- Calculate RMSE for past and future values

$$\left(\frac{\sum(\text{actual} - \text{predicted})^2}{n}\right)^{0.5}$$

pmts	acc	dev	cal	regr.rpart	regr.ranger	regr.xgboost	regr.xgboost_prev	regr.glm	regr.cv_glmnet
125,261,750	40	35	74	3,276,251,556	1,145,325,092	504,733,568	180,676,800	567,626,123	193,852,526
62,657,370	40	36	75	3,276,251,556	1,145,309,205	282,347,904	97,294,160	279,502,680	216,713,451
63,467,681	40	37	76	3,276,251,556	1,145,296,469	1,016,450,048	228,004,720	1,896,344,964	276,806,762
26,041,979	40	38	77	3,276,251,556	1,145,284,081	468,161,120	201,791,488	435,045,710	353,563,580
33,947,274	40	39	78	3,276,251,556	1,145,273,028	217,377,376	97,800,608	5,733,214,785	451,604,592
37,258,687	40	40	79	3,276,251,556	1,145,296,139	217,377,376	97,800,608	628,080,403	576,831,775

Results using RMSE

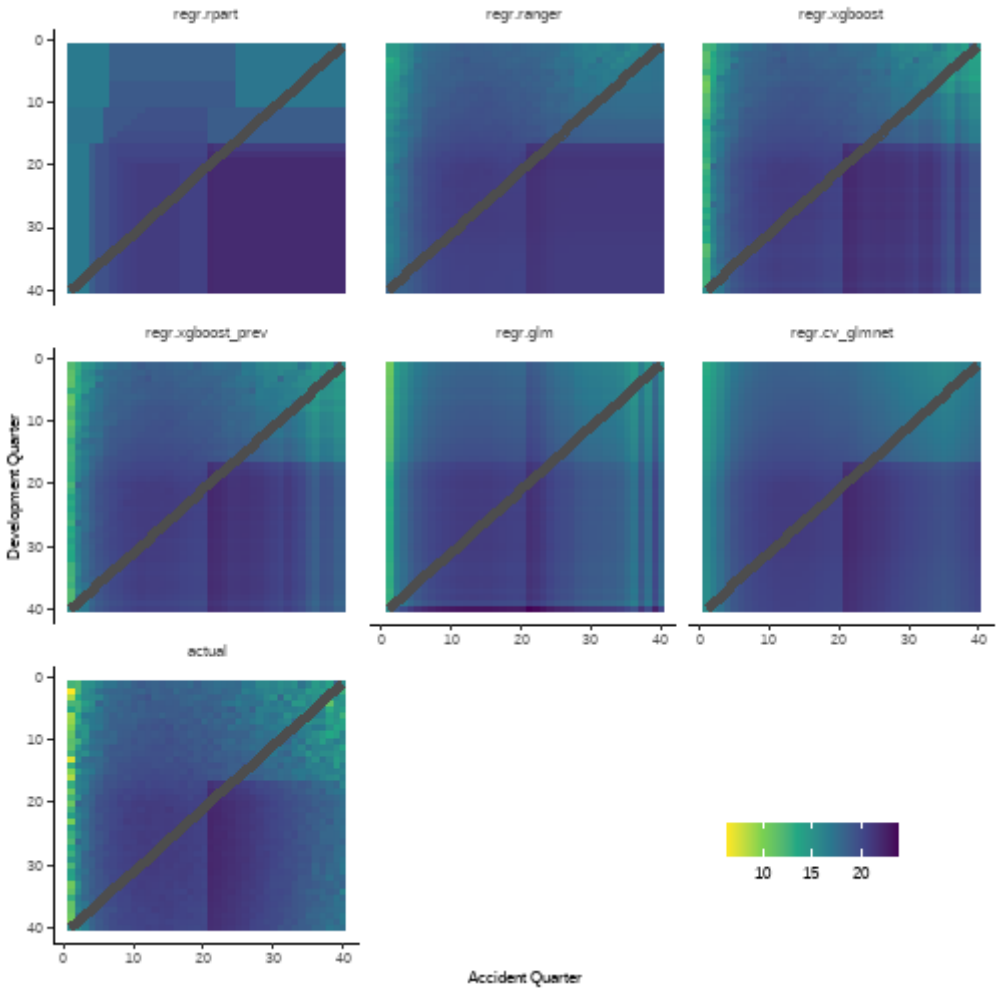
Past data

model	num	rmse
regr.xgboost_prev	820	30,264,935
regr.xgboost	820	32,006,128
regr.cv_glmnet	820	46,966,367
regr.rpart	820	63,506,568
regr.ranger	820	66,490,085
regr.glm	820	138,528,927

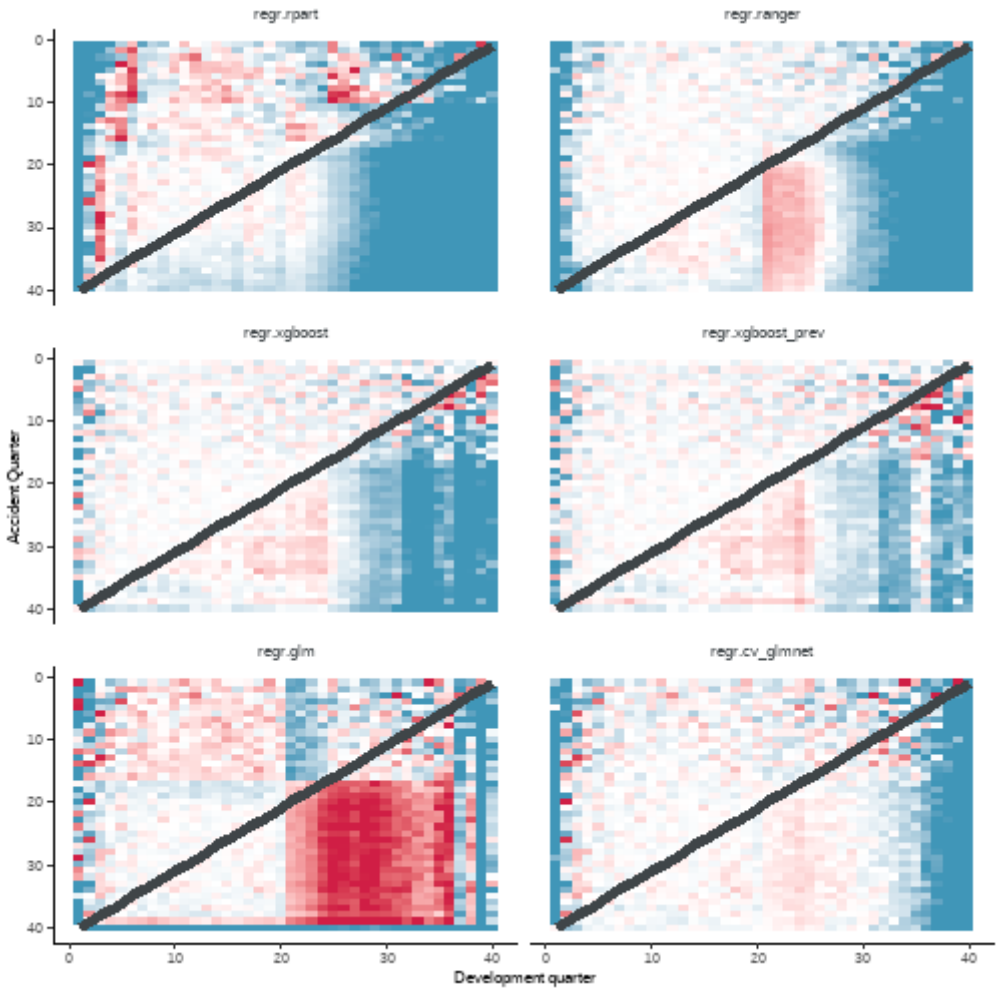
Future data

model	num	rmse
regr.cv_glmnet	780	251,388,945
regr.xgboost_prev	780	322,701,580
regr.xgboost	780	578,290,276
regr.ranger	780	820,612,713
regr.glm	780	1,722,472,390
regr.rpart	780	1,916,306,264

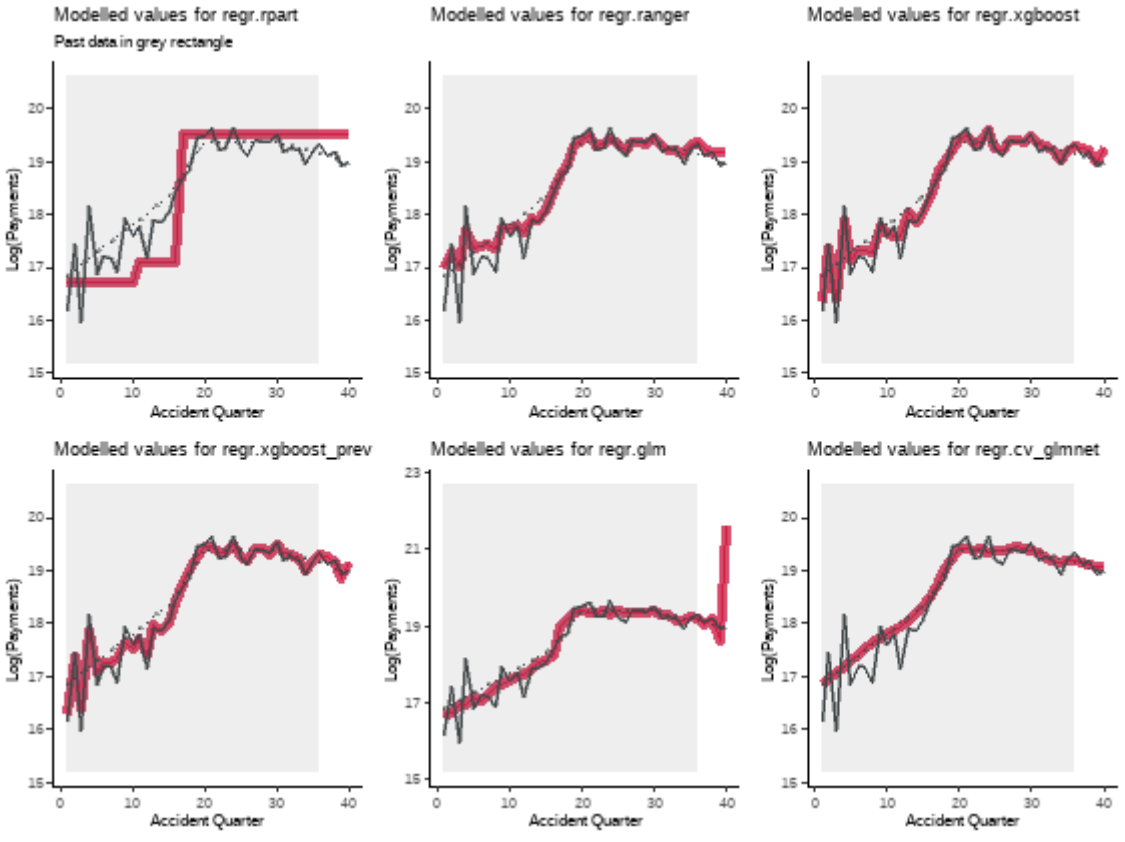
Fitted values



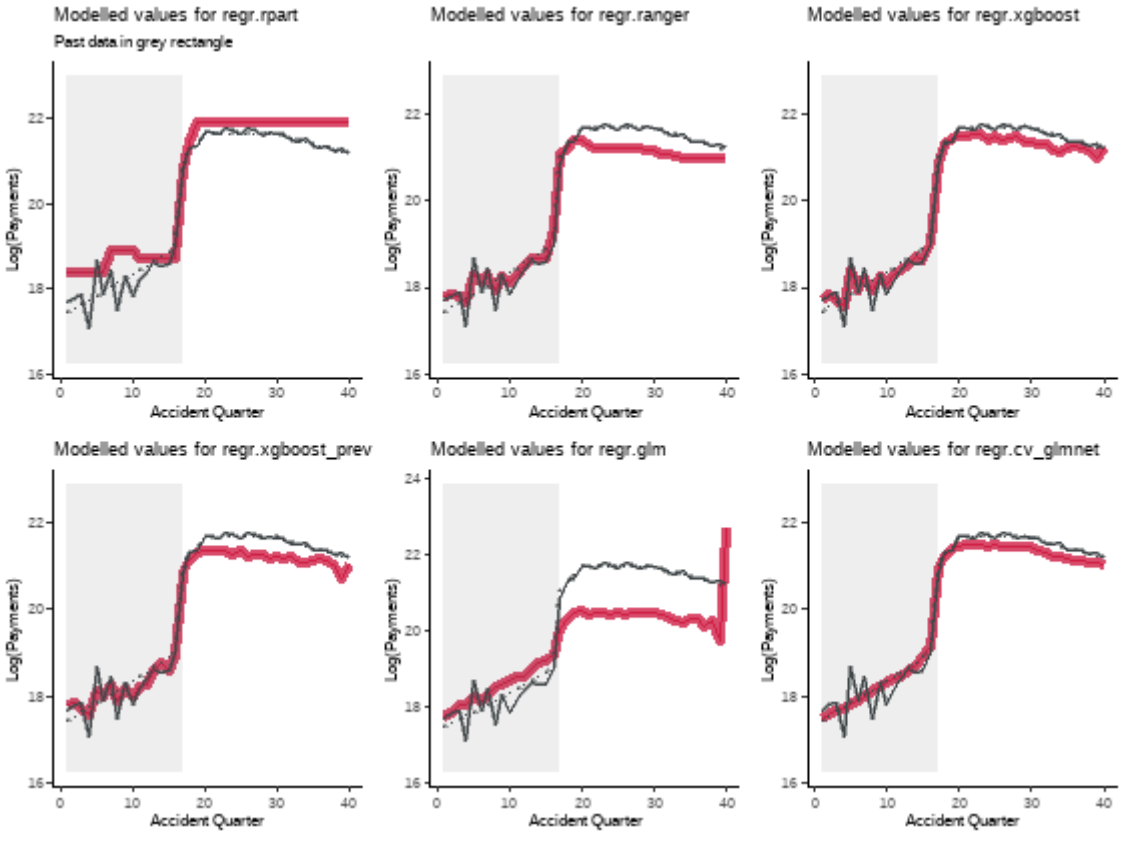
Heat maps of fitted values



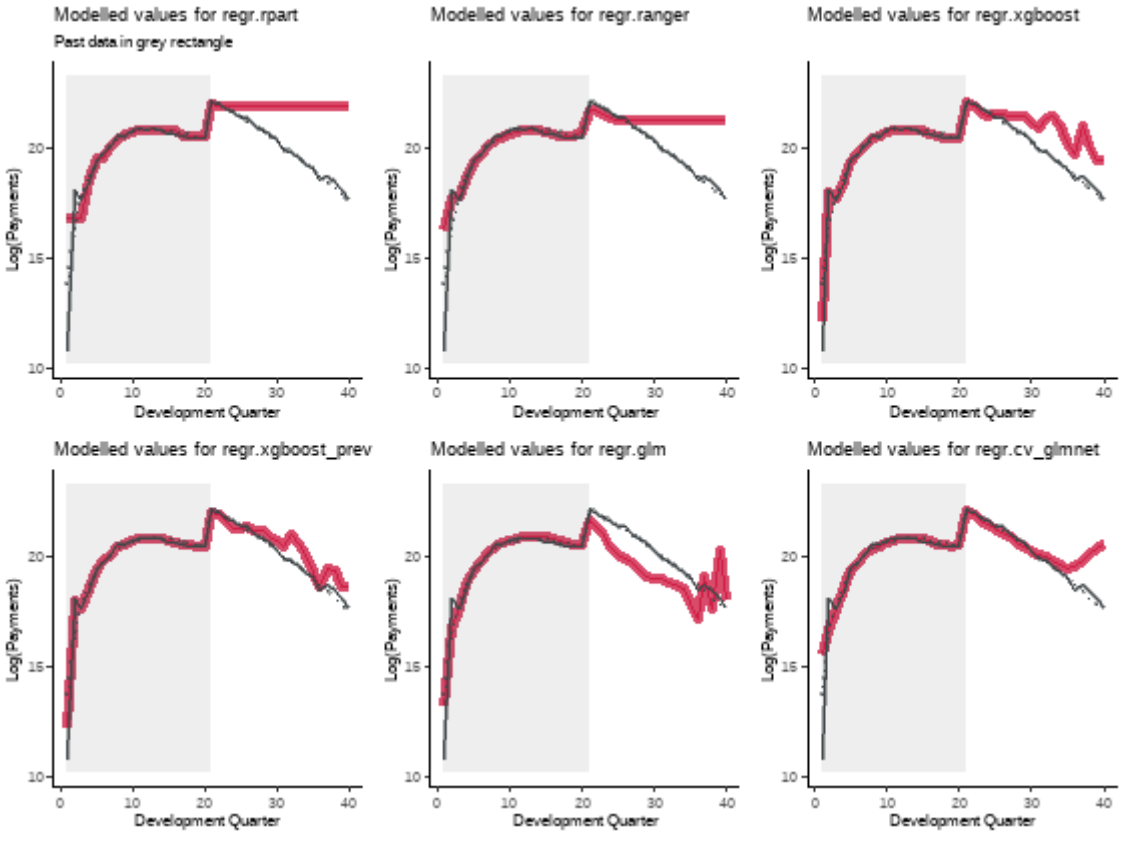
Tracking plots - acc for dev=5



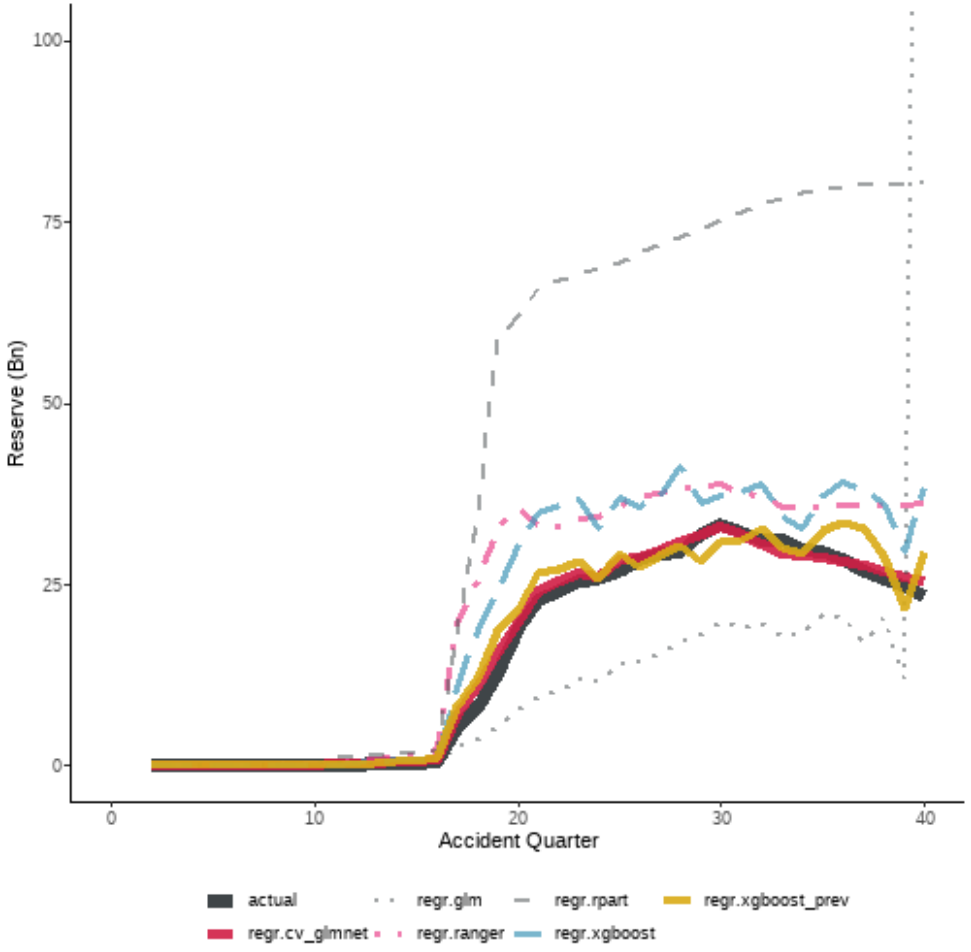
Tracking plots - acc for dev=24

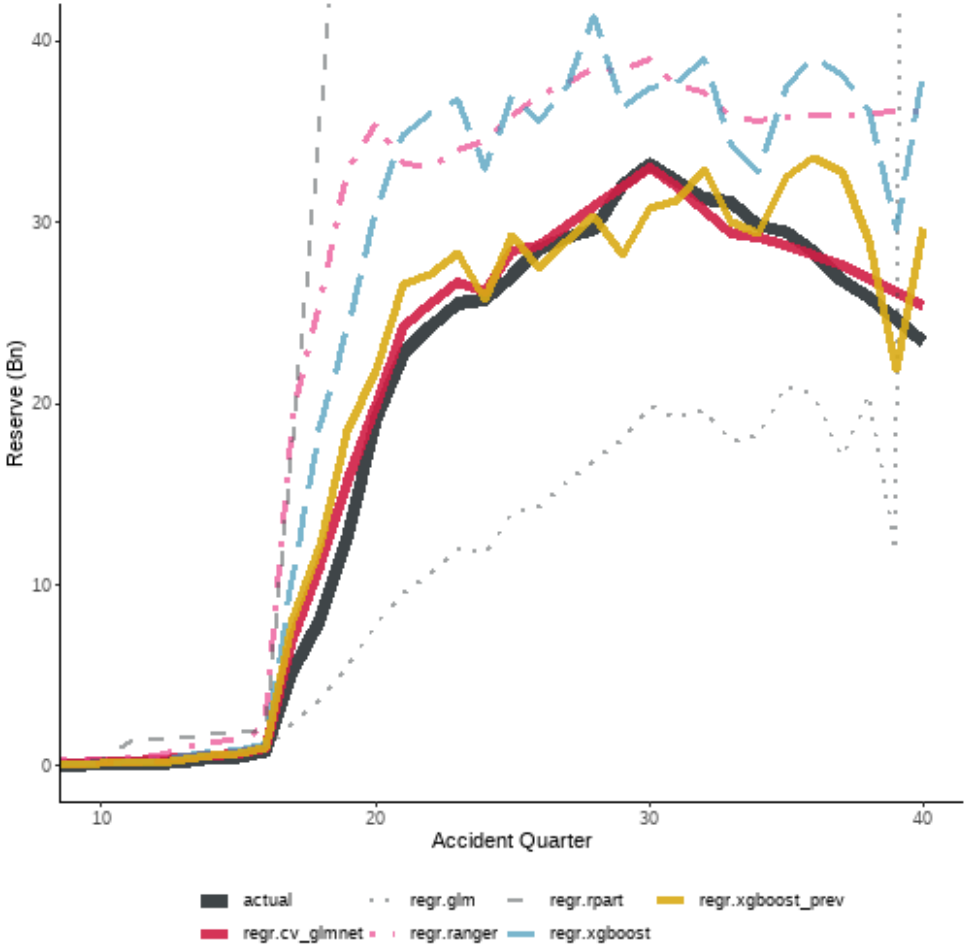


Tracking plots - dev for acc=20



Reserves


Reserve (Bn)
100
75
50
25
0
0
10
20
30
40
Accident Quarter
actual
regr.glm
regr.rpart
regr.xgboost_prev
regr.cv_glmnet
regr.ranger
regr.xgboost


Reserve (Bn)
40
30
20
10
0
10
20
30
40
Accident Quarter
actual
regr.glm
regr.rpart
regr.xgboost_prev
regr.cv_glmnet
regr.ranger
regr.xgboost

Reserves - amounts

Model	Reserve(Bn)	Ratio to actual(%)
actual	608	100
regr.cv_glmnet	628	103
regr.xgboost_prev	649	107
regr.glm	563	93
regr.xgboost	817	134
regr.ranger	845	139
regr.rpart	1,680	276

- The best performers are the LASSO and the previously tuned XGBoost model.
- The overall reserve for the Chainladder model (regr.glm) hides the fact that this result is actually significant under-estimation in most accident quarters balanced by significant over-estimation in the last.

7 - Conclusions and Commentary

Conclusions

Summary

- Worked example on a familiar (but tricky) data set
- Plug and played different models
- Model analysis tools
- The LASSO performs best for these data
  - but was developed specifically for this purpose
- Data set is small
  - Best performance of ML often requires big data

Observations

- No surprise that these are poor:
  - decision tree (rpart)
  - Chainladder
  - in practice, no actuary would use the Chainladder result without adjustment
- No real surprise that xgboost outperforms random forests (ranger)
- The two very different xgboost trees were a surprise
  - hyper-parameters are very similar
  - performance on future data is very different
- Did we just get lucky with the better performing version?

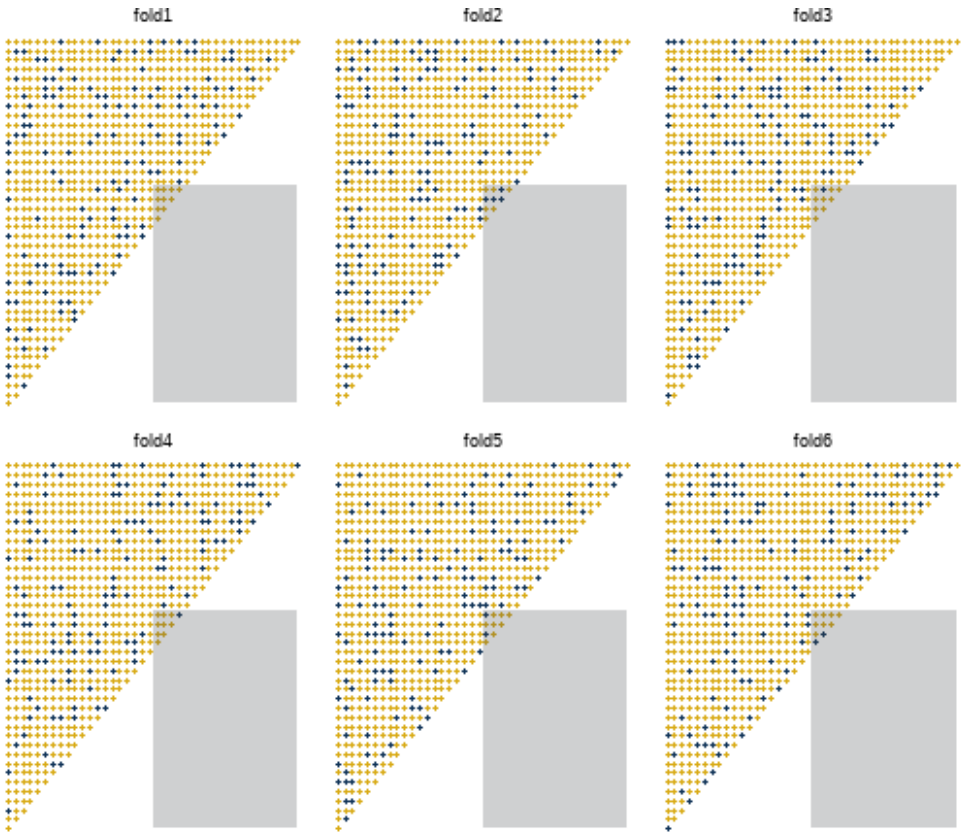
Model validation process

Comment 1

- Future data set is dominated by large interaction
  - but only 10 cells out of 820 in past data
  - hard to detect?
  - hard to know how far to project?
- Performance on future data largely determined by how well this interaction is captured.

Comment 2

- Standard cross-validation techniques may not be the best choice for a reserving problem
  - they look at performance within a data set
  - but reserving is a forecasting problem.
- Rolling origin cross-validation may be a better choice
  - see, e.g., Balona and Richman 2020



What next?

Explore this example

- Try running the code
  - R code
  - Python code
  - Use other data sets
- Check our website regularly for new material

Want something more challenging?

- Advanced XGboost reserving example (series of 3 posts)