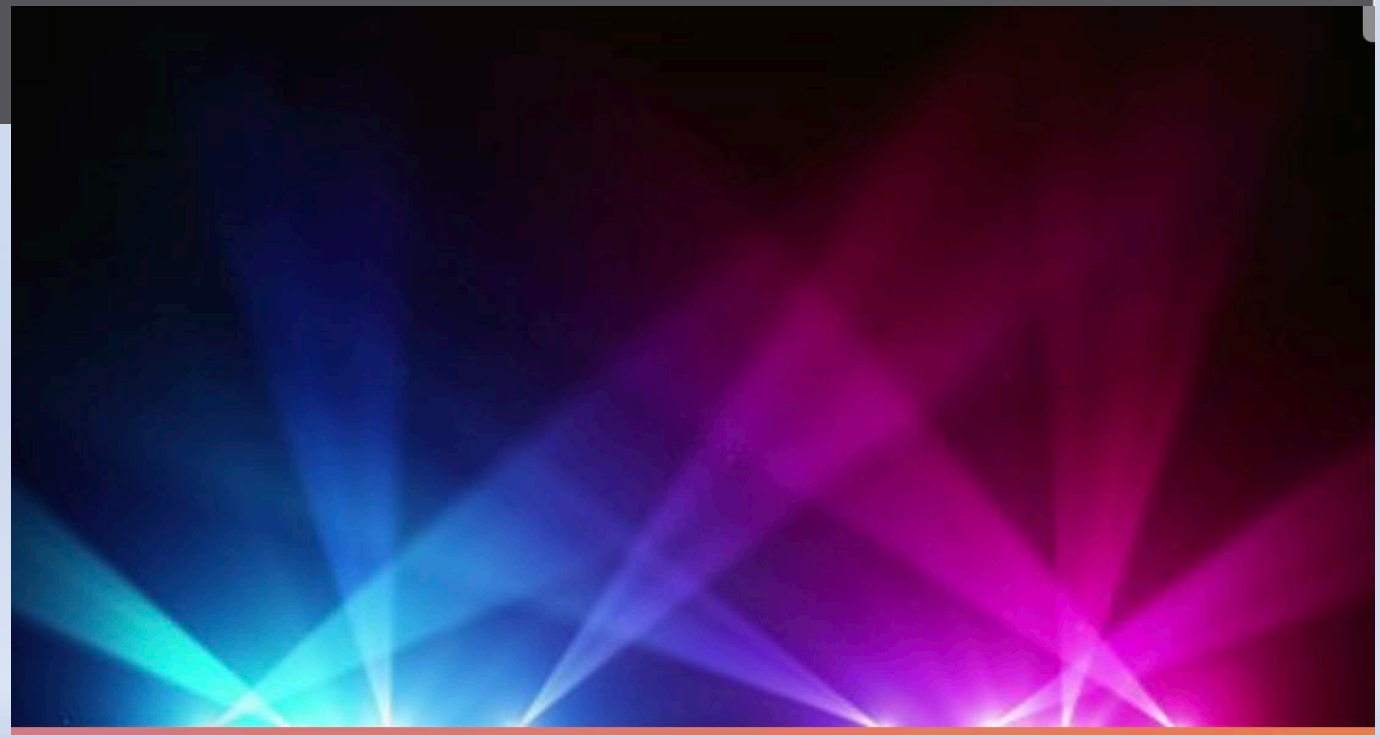
Emerging "Bayesian" Techniques in Application

CAS RPM 2021

17 March 2021

Gary Venter
Navarun Jain



Improve Model Predictive Accuracy with “Bayesian” Smoothing Splines

Gary Venter

Splines for Regression Models

- Fit curves across row parameters, column parameters, etc.
- Can get good models with fewer parameters
- But usually, parametric curves will not capture important effects
- Splines piece together curves out of components – linear or cubic
- Smoothing splines increase smoothness of the splines by imposing constraints, like minimizing a selected λ times a roughness measure
- E.g., the sum of squared 2^{nd} or 3^{rd} differences of the curve:
  - *Whittaker, E.T. (1922). "On a new method of graduation". Proceedings of the Edinburgh Mathematical Society.* ***41****: 63–75.*
- For cubic splines, the integral over the continuous curve of its squared 2^{nd} derivative has become a popular roughness measure
- For linear splines, the sum of the squared 2^{nd} differences is comparable

Why Smoothing?

- Improves accuracy: reduces predictive and estimation variances
- Comes out of actuarial credibility theory – reduce prediction error by weighting against grand mean – started by Mowbray 1914 CAS paper
- Stein 1956 showed that for any estimate of 3 or more means, weighting against grand mean to some extent reduces error
- Buhlmann credibility is same as James-Stein estimator in simple cases
- 1970 paper brought this to regression by shrinking coefficients:
    - Hoerl, A.E., and R. Kennard. 1970. “Ridge Regression: Biased Estimation for Nonorthogonal Problems.” Technometrics 12: 55–67.
- Variables first standardized to be mean 0, variance 1, which shrinks fitted values to grand mean
- As in credibility, this reduces errors but biases estimates towards that mean

Smoothing Splines Use Shrinkage

- Ridge regression minimizes NLL + $\lambda \sum \beta_j^2$
- Smoothing splines minimize NLL + $\lambda \sum 2nd\ dif_j{}^2$ for linear splines or NLL + $\lambda \int f''(x)^2$ for cubic splines
- Lasso minimizes NLL + $\lambda \sum |\beta_j|$. Can do that for splines also.
- Lasso popular because some coefficients shrink to exactly zero, eliminating those variables. So used for variable selection as well.
- Problem with all of these, including credibility, is choosing λ.
- Usually done with cross validation: leave out maybe 10% of the observations in each of 10 fits, and measure $\sum$NLL of left out points
- There are problems with that, discussed below

“Bayesian” Version of Shrinkage

- Give the variables shrinkage priors: mode zero priors
- The probability is greatest around zero so parameters closer to zero are favored, but further away is ok if it improves the fit enough
- E.g., normal priors, double exponential, t-distributions. σ^2, etc. tell how much deviation from zero is likely – so controls shrinkage like λ.
- Estimate by MCMC: Markov Chain Monte Carlo estimation
- It’s a collection of numeric methods for sampling from a product of two distributions, here the joint likelihood: likelihood times the prior.
- Gives a sample of the distribution of the parameters given the data
- Measures fit by leave-one-out cross validation: every point is used as a left-out subsample
- Loo measure can be quickly estimated from the sample of posterior fits. Weighting samples more that fit poorly at a point produces a good weighted estimate of the likelihood of that point had it been left out.

Not really all that Bayesian

- Priors are not subjective but are part of the postulated model, like residual distributions are, and can be tested by model performance
- Parameters here are basically frequentist random effects – frequentist parameters are fixed points with estimation variances, but random effects are like Bayesian parameters and have postulated distributions
- What Bayesians call posterior distributions are also the conditional distributions of the effects given the data
- With normal prior, the parameter NLLs ~ $\beta_j{}^2$, so log joint probabilities ~ data NLL + $\Sigma\beta_j{}^2$ and mode gives ridge regression
- With double exponential prior, the parameter NLLs ~ $|\beta_j|$ so log joint probabilities ~ data NLL + $\Sigma|\beta_j|$ and mode gives lasso. This prior is called Bayesian lasso

Cross Validation and Goodness of Fit

- Penalized likelihood measures like the AIC aim to correct for sample bias in NLL as a fit measure. NLL is overstated if measured on the sample that the parameters were fit to. This bias is greater if more parameters are used, so NLL is penalized by a multiple of number of parameters.
- With shrinkage estimates like lasso and smoothing splines, shrunk parameters do not use as many degrees of freedom and the more shrinkage there is, the fewer effective parameters, so AIC etc. don't work.
- NLL from cross validation gives another estimate of sample bias – but no set way on how many left-out subsamples to use, etc.
- Good for comparing fits across models but problematic to estimate λ or shrinkage σ^2 by optimizing cross-validation NLL: likely that best λ will be one that most under-estimates sample bias.
- "Bayesian" approach gets around this by letting you put a prior on σ^2 and use its conditional expected value given the data as the estimate

Bayesian Version of Smoothing Splines

- For linear splines, can make the parameters be the 2nd differences of the fitted points on the spline curves. Every later 1st difference has that parameter in it and gets added to every later spline point as well. Thus, it gets added to each subsequent point one more time.
- For the variable for row j, an observation from row z gets $[1+z-j]_+$ times the parameter for that variable, so the dummy for the variable is $[1+z-j]_+$ at that observation. The same holds for columns, diagonals,… (here z could be a real number used to interpolate the spline or at the observed points)
- This design matrix can be used in straight regression, lasso, or Bayesian estimation for the 2nd difference (slope change) parameters.
- Cubic is with slight adjustment of (also see links from there): https://stats.stackexchange.com/questions/172217/whyare-the-basis-functions-for-natural-cubic-splines-expressed-as-they-are-es

Cubic Splines

- Cubic splines have design matrices as well that can be used in straight regression, or in constrained regression that limits a roughness measure, like the squared 2nd derivative or the 2nd or 3rd differences.
- For n rows there are n variables. The 1st variable is the constant term and is 1 for all observations. The 2nd is the linear term and = z for an observation from row z.
- After that, for $z \leq n - 1$, the jth variable dummy value is $[2+z-j]^3_+$.
- For $n - 1 < z \leq n$, the jth variable is $[2+z-j]^3_+ - (2+n-j)[1+z-j]^3_+$.
- These assume the spline is linear for z ≤ 1 and z ≥ n.
- The fitted values on the spline are the parameters β times design matrix, and the integral of the squared 2nd derivative is a closed form formula.
- Increasing the λ shrinkage factor reduces the βs except for the constant and linear term. This can be closely approximated by using lasso or ridge shrinkage priors on the βs instead, which can be done in Excel or MCMC.

Example fit to workers comp 15x15 triangle

- Can fit smoothing splines to every dimension in a rectangle or higher dimension version, like in class pricing
- Or to loss development and reserving fits to triangles, i.e., parts of rectangles
- Can fit them to diagonal parameters as well
- Whole data set is strung out into a single long column, keeping track of the row, column and diagonal each observation comes from
- Spline dummy variables are created for each row, column, and diagonal parameter. These depend on row, column etc. each point comes from
- Usually assume row, column, etc. factors multiply so often start by modeling log of losses as a linear regression in the factors

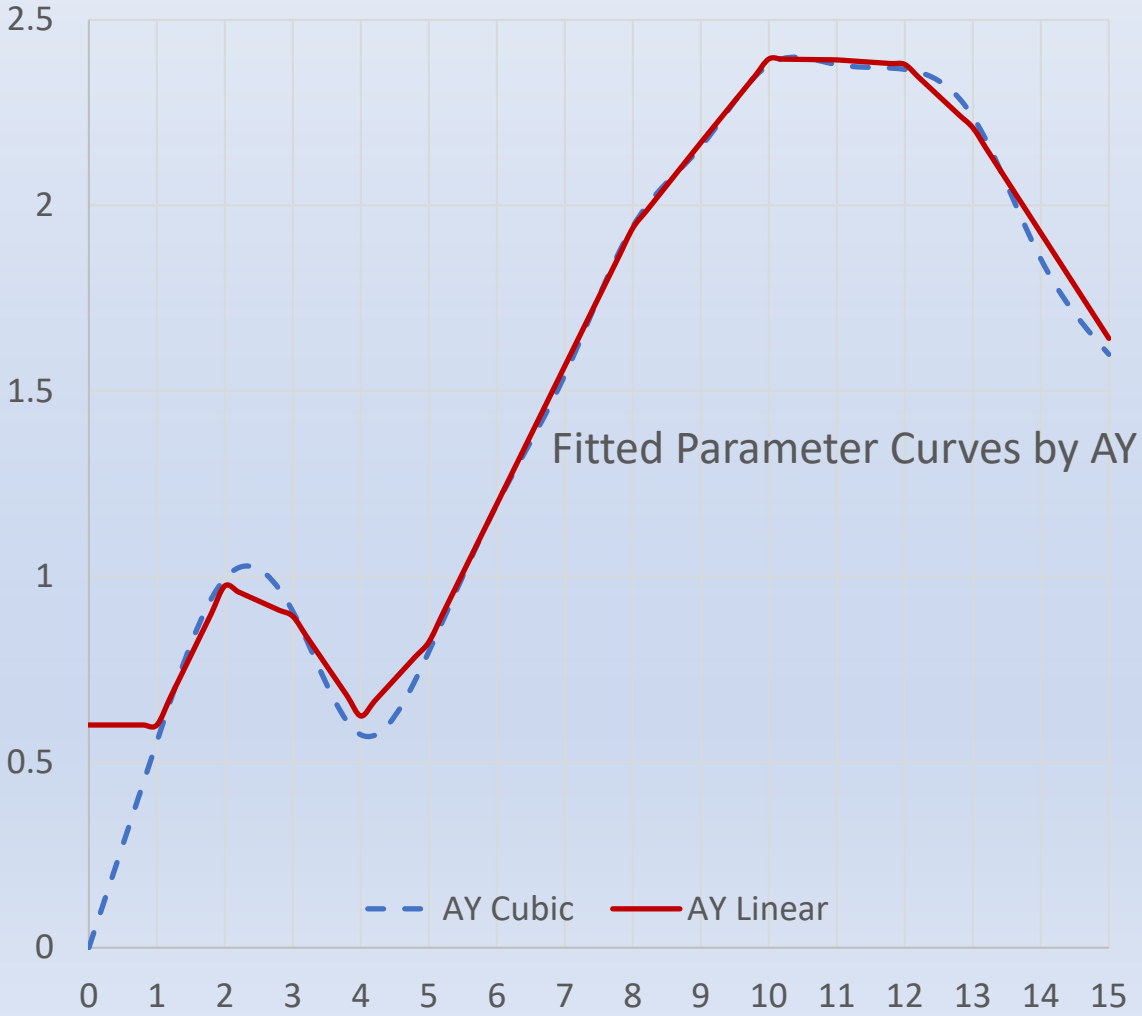
Work Comp Cumulative Paid Triangle

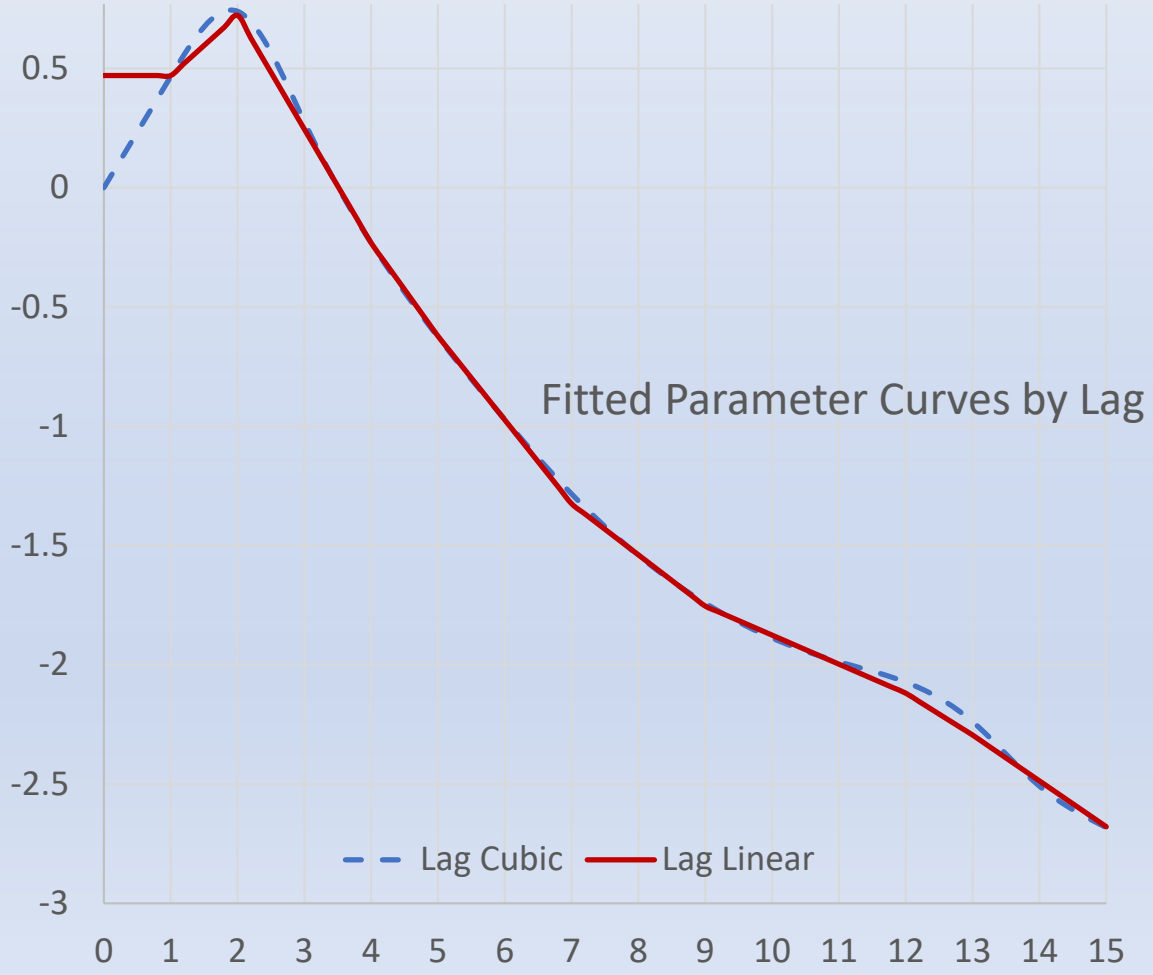
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	129124	217397	118421	63984	39196	31450	19809	14556	8420	8507	7319	8937	10274	5383	5594
2	216427	325705	171013	92336	65834	41287	28358	22541	15742	16934	12498	17058	10786	7834	0
3	190698	207945	132229	74989	41989	32347	26640	21860	26870	17868	15149	15073	10549	0	0
4	117415	166555	95082	55577	40785	28737	16396	15102	12993	16123	10323	8979	0	0	0
5	133903	196230	110738	68289	52485	31817	23922	25356	17653	13431	14458	0	0	0	0
6	201997	289239	165234	103281	70280	57042	39964	34252	25683	21589	0	0	0	0	0
7	293284	435794	242665	159174	108122	75959	53191	42359	35453	0	0	0	0	0	0
8	409026	605225	381820	235599	162399	113710	81298	70265	0	0	0	0	0	0	0
9	514682	799168	437650	281391	203616	143531	109599	0	0	0	0	0	0	0	0
10	674265	886002	512631	364672	263383	215784	0	0	0	0	0	0	0	0	0
11	692986	905049	546113	396918	266963	0	0	0	0	0	0	0	0	0	0
12	700094	929014	590184	366655	0	0	0	0	0	0	0	0	0	0	0
13	627820	834302	517097	0	0	0	0	0	0	0	0	0	0	0	0
14	449942	552715	0	0	0	0	0	0	0	0	0	0	0	0	0
15	326810	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fitting Plan in Excel – See Spreadsheet

- Make a long column for log of losses and use just AY and lag variables
- Build linear and cubic spline design matrices leaving out 1st AY and 1st lag in linear splines and using only 1 constant variable in each matrix.
- Start with a regression with no shrinkage for each matrix. With design matrix x and log loss variable y, the parameters are $(x'x)^{-1}x'y$, which can be calculated in Excel with matrix mult, inverse, transpose functions.
- For linear and cubic splines, these regressions agree on the fitted values, and these are the same as those from using 0,1 dummies.
- Shrink βs by lasso with Excel's solver, keeping the regression parameters as starting values. Begin with λ=0.00001 and gradually increase it by factors of 10, using the answers from the previous λ as starting values. Keep going until there is some noticeable change in parameters.

Lasso splines, with λ = 0.1 (Graphs interpolated) Linear or cubic had 9 or 8 (of 28) |β| < 0.001 and 11 or 16 < 0.01. Normal σ□s = 0.131 up from 0.122 @ λ = 0





MCMC Version

- Took out variables with lasso parameters $|\beta| < 0.001$
- Used double-exponential (Bayesian lasso) priors in 0,s for parameters (except the constant), with log of scale parameter s uniform on [-5, 5].
- This came out E(s) = 0.2414 for linear and = 0.1563 for cubic. Ranges [0.025, 0.975] of log posteriors were [-1.90,0.97] and [-2.39,-1.38].
- More shrinkage for cubic, as design dummy variables have higher values, with the cubes and all, and so parameters are smaller.
- Loo out-of-sample cross validation LL was 55.5 for linear and 53.9 for cubic, so linear gave a slightly better fit. Residual σ□s 0.139 for linear and 0.141 for cubic, so these had more shrinkage than $\lambda = 0.1$ lasso.
- Function loo_compare found the difference of 1.5 in loo had a standard deviation of 2.3, so really not significant

Distribution Assumptions

- Log regression can also be used for log of a parameter in the assumed distribution of losses, so exponentiating gives that parameter
- For instance, for losses gamma in a_j, b_j, you might assume every cell has the same b parameter, so the mean is $a_j b$ and the variance is $a_j b^2$. Then variance/mean = b is constant across the cells. This often works well for total loss $.
- Models for loss severity often assume all the cells have the same a parameter, so the mean is ab_j and the variance is ab_j^2. Then CV^2 = variance/mean = 1/a is constant. Log regression gives lognormal for losses with CV^2 a constant (= function of fitted σ^2)
- Also could start by fitting lasso with the R glmnet app with cross validation for λ, which will show parameters that can be set to 0, so the variable for those factors just left out of MCMC. With spline models, that just keeps previous curve going at the left-out point.

Gamma Spline Fits in Excel

- No closed-form solutions like the regression we used for $\lambda = 0$ in the log of losses model. But solver needs good starting parameters.
- Log lasso fits give parameters for lognormal fit to losses. So use those means μ_j as a starting point for gamma fit.
- The gamma mean is $\mu_j = a_j b$, so make b a parameter (replacing σ) and assume that $loss_j$ is gamma in μ_j/b, b.
- We need a fairly good b to start with. So in Excel, maybe compute all the fitted likelihoods as gamma.dist($loss_j$, μ_j/b, b, 0), (the last 0 denotes density) and try b = exp(10, 11, 12, …) one at a time until at least the likelihoods all come out non-zero in Excel.
- Then do the same lasso estimation at $\lambda = 0.1$ we used in log model

Gamma Spline Fits in Excel

- This worked fine for the linear splines. The function being minimized, NLL + $\Sigma|\beta_j|$, came to 1306.
- There was a problem for cubic splines. Some of the variables have parameter distributions that are near cliffs. That is, in one direction or another, anything but the very smallest change in the parameter will produce a huge increase the the function being minimized. The algorithms built into solver just cannot handle that. Even the R glmnet function has problems with lasso for cubic splines. MCMC has more complicated search algorithms and can deal with it, but it takes them a longer time.
- Minimizing over some of the parameters gave a fit of 1316, which is not bad but probably is not optimal either.

Ridge Regression for Splines Easy in Excel

- Has closed form parameters, just like regression. Minimizes NLL + λ[sum of squares of spline parameters]. Usually minimize linear factor of cubic, in case it is not too important to the fit, even though it is not in 2nd deriv.
- Regression parameters are $\beta = (x'x)^{-1}x'y$, and in any ridge regression with λ, $\beta = (x'x+\lambda I)^{-1}x'y$, where I is the $n_x n$ identity matrix.
- Can use in Excel for cubic and linear splines for any log regression, even with diagonal parameters. Just take out 2 or 3 row/column variables first for identifiability of all parameters – take ones with low parameters.
- To approximate lasso, this can be iterated by $(x'x+\lambda I\beta^{-1})^{-1}x'y$, with β^{-1} the reciprocals of β
- Tried this for cubic splines for one iteration in spreadsheet – it's getting towards lasso parameters. If some parameters are too close to zero, this can get distorted, and those can just be taken out instead (set to zero). Will get to lasso eventually.

MCMC Fits of Splines to Gamma Losses

- Same formulas in spreadsheet done in R Stan package with Laplace = double-exponential priors for βs.
- For cubic needed starting parameters. Used ridge regression estimates.
- Loo = -1283.2 for linear and -1285.9 for cubic.
- Loo_compare says difference of 2.6 has sd = 2.6, so not very significant
- Loglikelihood was -1270.2 for linear and -1271.6 for cubic, so also better
- Mean of Laplace scale s was 0.2377 for linear and 0.1481 for cubic, similar to log spline models

Extensions

- Easy to add diagonal (year-of-payment trend) parameters, as above, get rid of unnecessary ones, and compare by loo to see if they help
- Trend impact could vary by lag. In comp, this could be from a medical inflation trend, as medical is a bigger and bigger part of later payments
- In liability it could relate to delays from the legal system
- Another constraint is needed on the trend weights. If the raw weights by lag are in the vector r, set m = max(r) and p = exp(r–m), so p∈(0,1].
- Also make x the spline design matrix for rows and columns with parameters v and t the design matrix for diagonal with parameters u. Then model:
- $\mu = \exp(c + xv + p \times tu)$, say for the gamma model
- This did improve the fit to the comp triangle, with p increasing by lag

Conclusion

- It is easy to fit linear and cubic smoothing splines to variables in pricing and reserving regression models.
- Smoothness measure used is sum of squares or absolute values of the spline-curve parameters. This is the same for linear splines as measuring by 2^{nd} differences and gives very similar cubic splines to those from using the 2^{nd} derivative as the smoothing measure.
- For spline regressions for log of losses, this is a simple, closed form formula in Excel. For exponential link, this can be done by solver for linear splines, but seems to require MCMC software for cubic splines.
- Even in MCMC, linear splines fit much faster. Results are similar for both linear and cubic splines.
- So far, no clear benefit seen for cubic over linear, except maybe if you want to interpolate the curves

LUX
ACTUARIES & CONSULTANTS

COMBINING BAYESIAN INFERENCE WITH MACHINE LEARNING:

*A (Brief) Intro to **Bayesian Neural Networks***

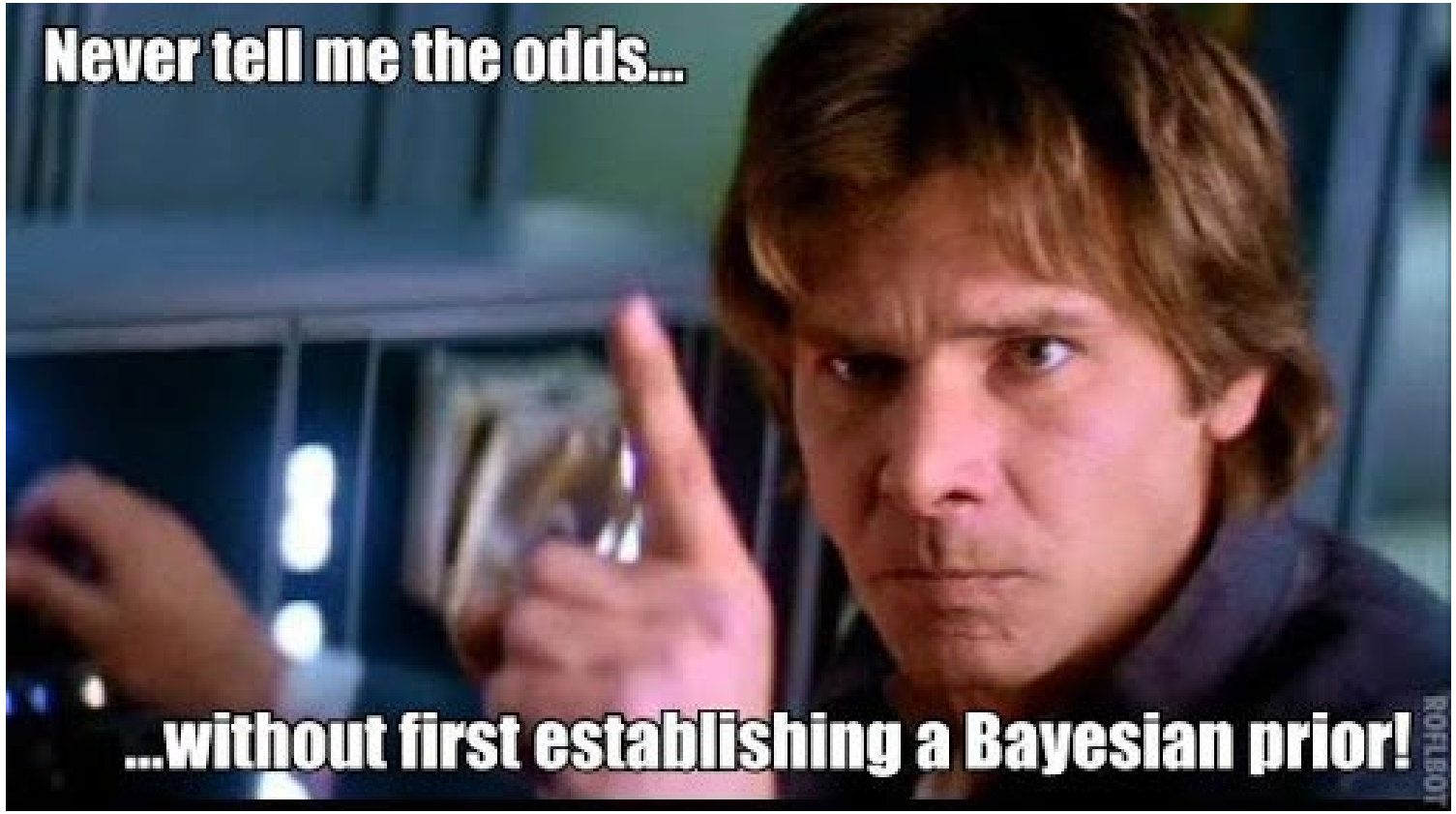
CAS RPM 2021

17 March 2021

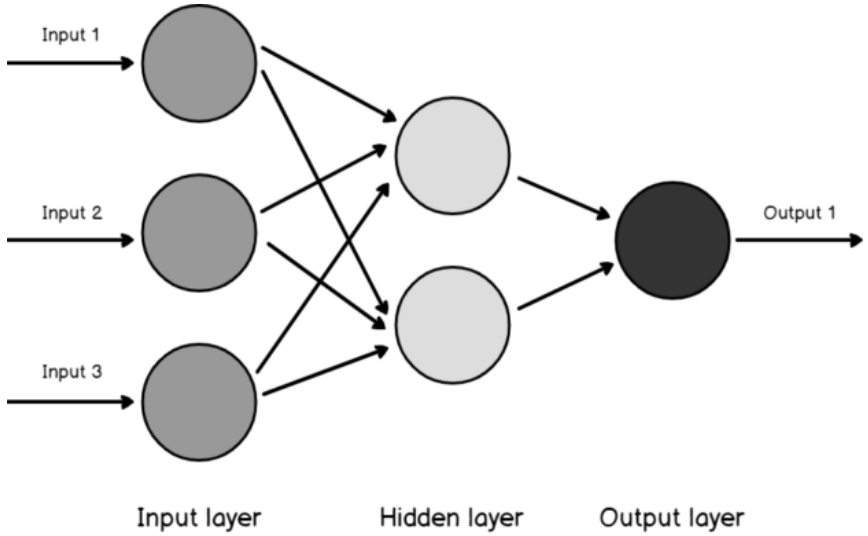
Navarun Jain
Actuarial Analyst
Lux Actuaries & Consultants

Agenda

- Intro to Neural Networks
- Extending the Framework – Going Bayesian
- Potential Uses
- Practical Application
- Key Takeaways and Conclusions


Never tell me the odds...
...without first establishing a Bayesian prior!
ROFLBOT

Intro to Neural Networks



1. Set random weights for all neurons
2. Compute network
3. Compare predictions to actuals based on loss function
4. Update weights based on optimization rule
5. Lather, rinse, repeat!

Structured: A Neural Network has a defined structure that consists of 3 types of layers

Sequential: Information flows in a sequence from one layer to the next, undergoing operations at each layer – almost like an assembly line

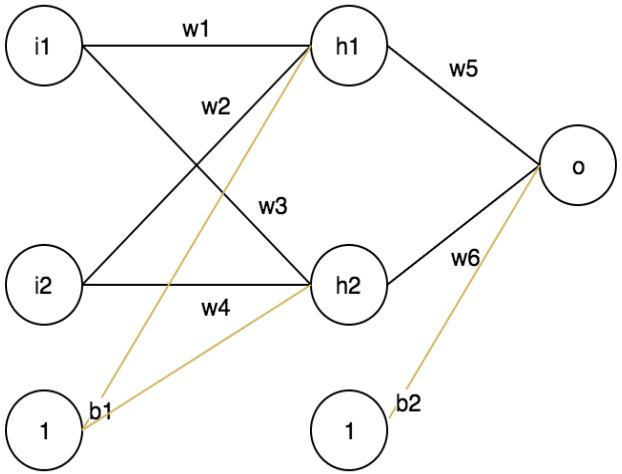
Intro to Neural Networks

- Cybenko (1989) first showed neural networks can act as *universal function approximators*
- Any approximation would have a degree of uncertainty
- Actuaries love uncertainty!
- Is there a way to capture uncertainty in neural network estimates?
- If so, what parameters to base uncertainty on?

Extending the Framework

Need to capture uncertainty in estimates – this is where *Bayesian Inference* comes in!

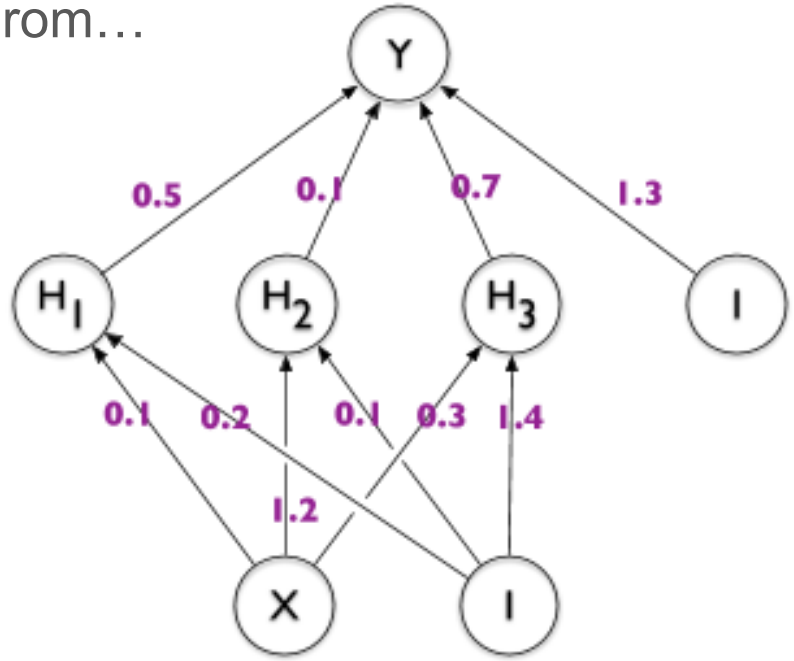
Extending the Framework



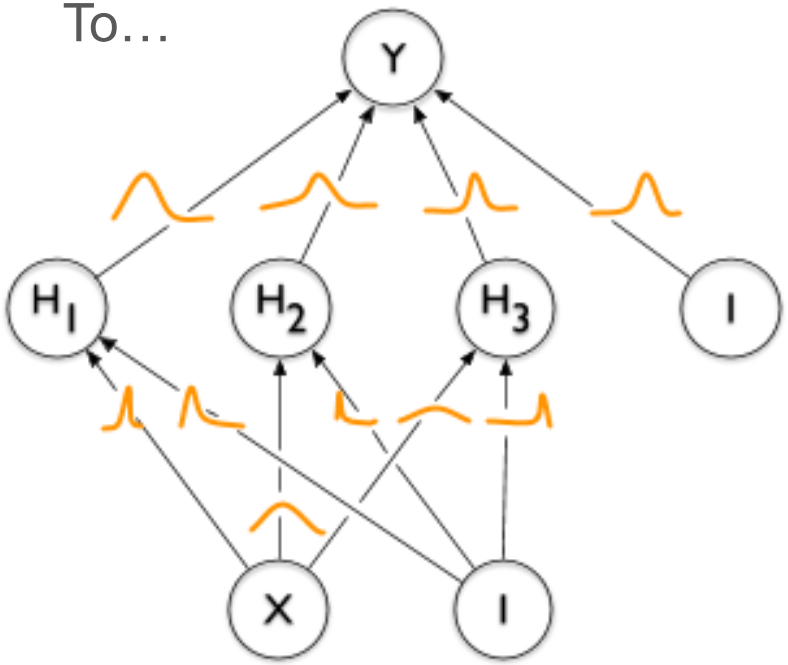
- Neural networks are parameterized by their *weights*
- Standard methods designed to learn point estimates of weights
- Bayesian Neural Networks estimate *distributions* of weights

Extending the Framework

From…



To…



This is the key idea behind Bayesian Neural Networks (BNNs)

Potential (Actuarial) Uses of BNNs

- Fraud Detection and Analysis
  - Quantify uncertainty in likelihoods of fraud – can help improve decision-making on which claims to investigate
- Pricing
  - Can use BNNs to generate and select more prudent estimates of claim frequency/severity in the absence of credible data
- Reserve Uncertainty Analysis
  - BNNs can potentially be fitted to claims triangles to generate distributions of reserves
  - Some initial research done on theoretically fitting BNNs to temporal data, yet to be fully explored
- Asset Liability Management & Capital Modelling
  - Results from BNNs can potentially be applied to frameworks like Solvency II which are based on Values-at-Risk

Practical Application

- Fraud Detection and Analysis
  - Quantify uncertainty in likelihoods of fraud – can help improve decision-making on which claims to investigate
- Pricing
  - Can use BNNs to generate and select more prudent estimates of claim frequency/severity in the absence of credible data
- Reserve Uncertainty Analysis
  - BNNs can potentially be fitted to claims triangles to generate distributions of reserves
  - Some initial research done on theoretically fitting BNNs to temporal data, yet to be fully explored
- Asset Liability Management & Capital Modelling
  - Results from BNNs can potentially be applied to frameworks like Solvency II which are based on Values-at-Risk

Practical Application

Running a BNN using PyMC3 on a sample Motor Claims Dataset

Data has flag for whether a claim was classified as Fraud or not

Detailed information for each claim

~12k records

Data

- Claim-level information with an indicator for whether a claim was flagged as a fraud or not

DRIVER

- Age
- Marital Status
- Gender

VEHICLE

- Country of Make
- Price
- Body Type

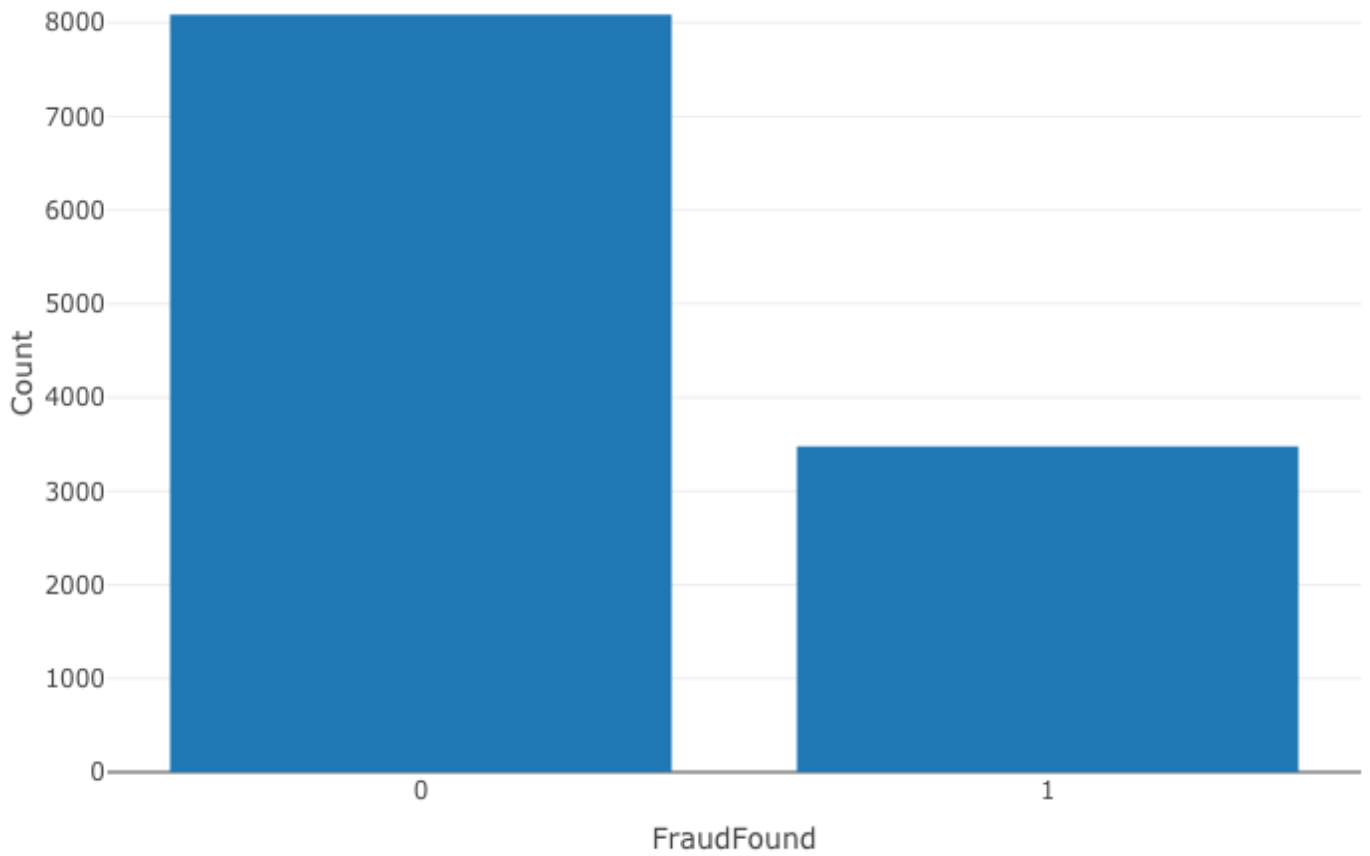
POLICY

- Cover Type
- # of Vehicles
- Deductible

SITUATION

- Claim File Date
- Witness Present
- Party at Fault
- Police Report Filed

Data


8000
7000
6000
5000
4000
3000
2000
1000
0
Count
0
1
FraudFound

The Model

```
def construct_nn(nn_input, nn_output):

    with pm.Model() as neural_network:

        nn_input = pm.Data('nn_input', X_nn)
        nn_output = pm.Data('nn_output', Y)

        w1 = pm.Normal('w1', 0, sigma = 1, shape = (X_nn.shape[1], 50))
        act_1 = pm.math.tanh(pm.math.dot(nn_input, w1))

        w2 = pm.Normal('w2', 0, sigma = 1, shape = (50, 20))
        act_2 = pm.math.tanh(pm.math.dot(act_1, w2))

        w_out = pm.Normal('w_out', 0, sigma = 1, shape = (20, ))
        act_out = pm.math.sigmoid(pm.math.dot(act_2, w_out))

        out = pm.Bernoulli('output', act_out, observed = nn_output, total_size = Y.shape[0])
    return neural_network

bmlp1 = construct_nn(X_nn, Y)
```

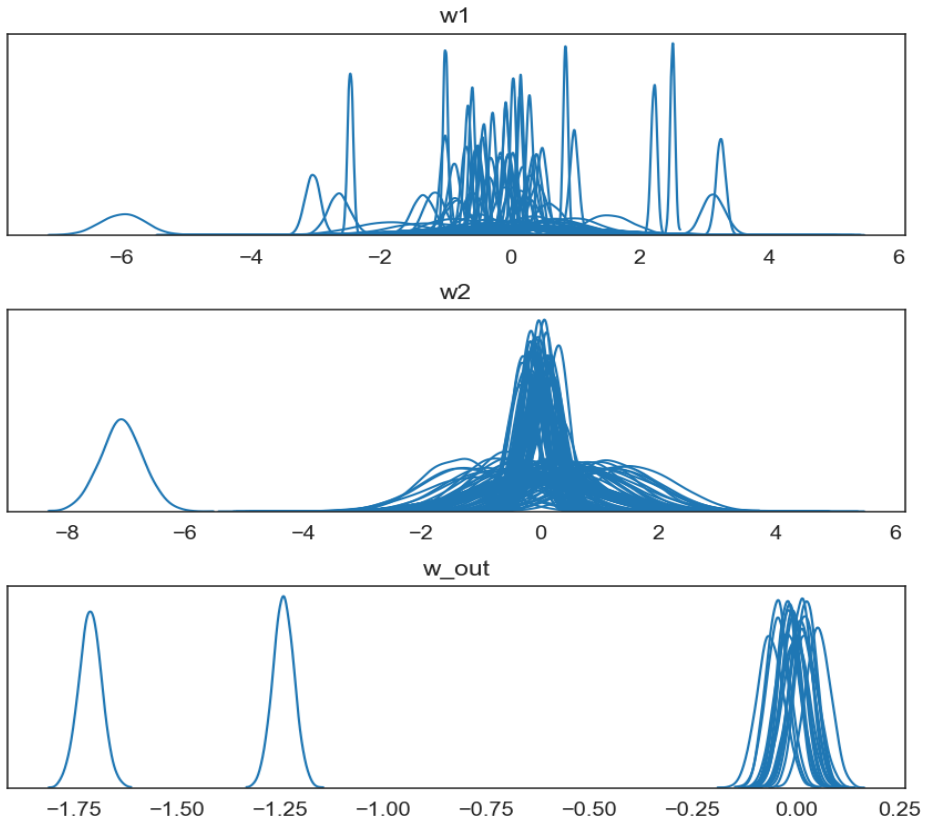
(50-20)
HL Configuration

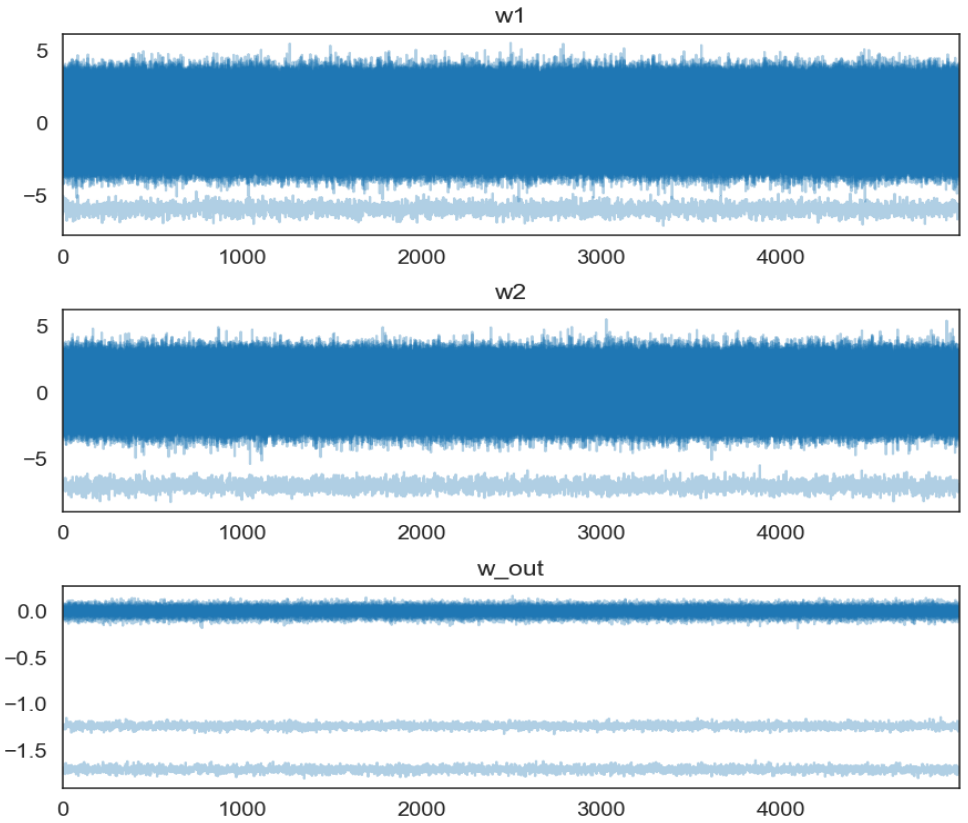
Gaussian (0,1) Priors

Bernoulli Likelihood

Posterior Distribution of Weights

Essentially just shifting Gaussians around!



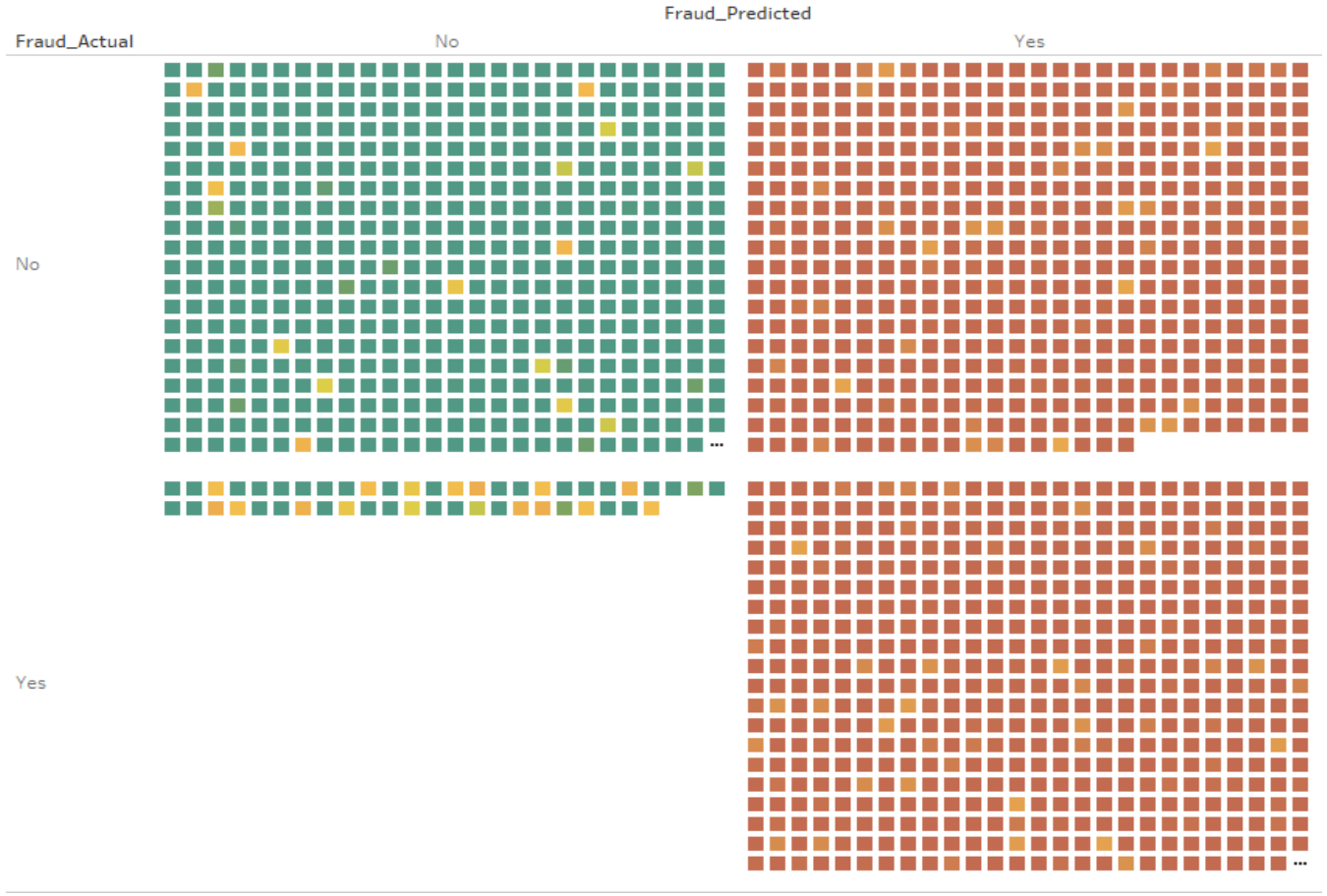


Predicted Fraud

From Point Estimates…

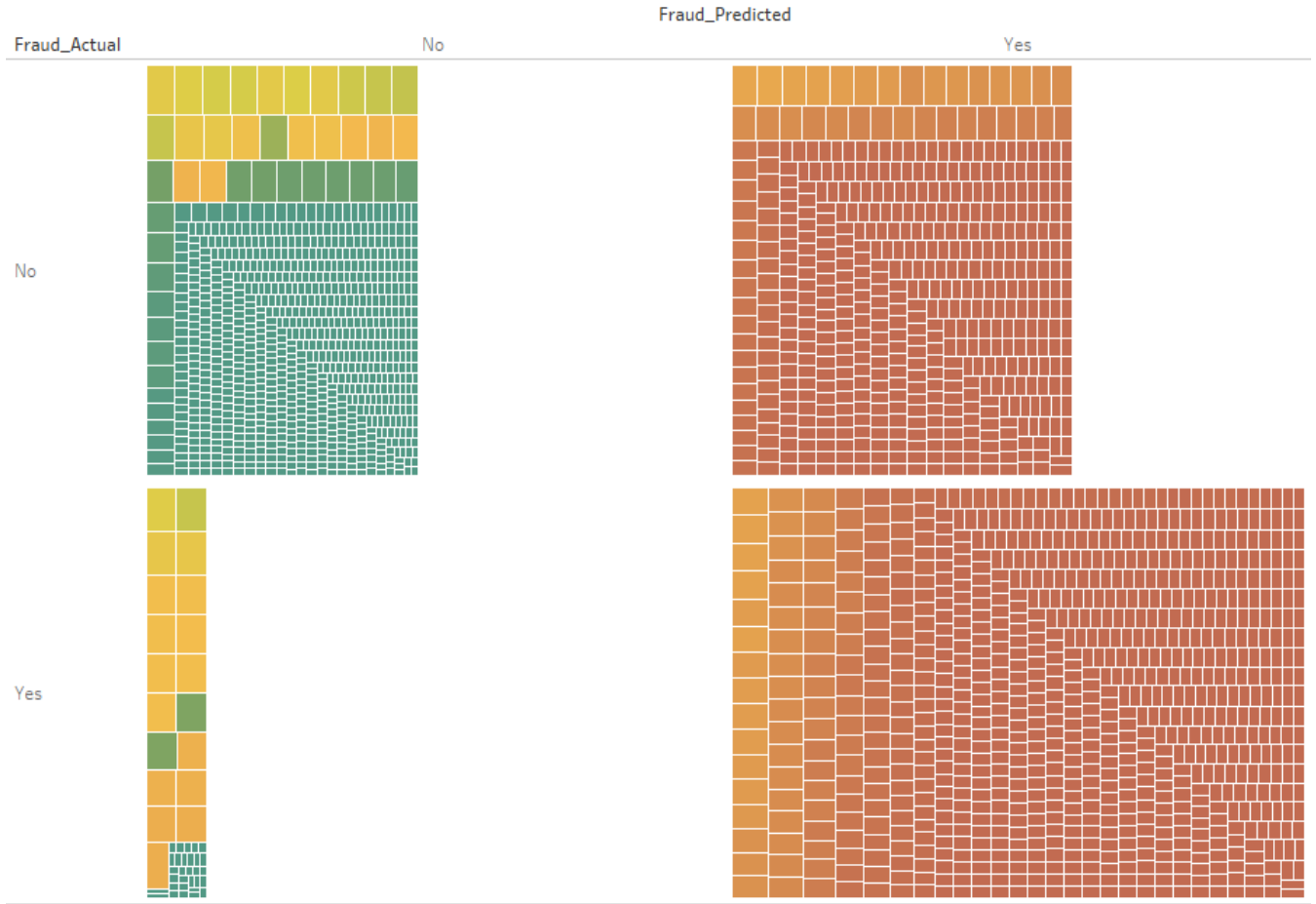
Predicted Fraud

… to Probabilistic Estimates…



Predicted Fraud

…with Uncertainty



The Good, the Bad & the Ugly

- BNNs can work better than normal methods for relatively small datasets
  - Exact inference methods work better than approximate inference
  - Exact inference methods are computationally more expensive to train
- Computational cost can be heavy during testing
  - Need to sample *n* times from posterior
- No specific rules for setting priors, mostly driven by assumptions

Key Takeaways

- Mixing Bayesian Inference with Machine Learning can unlock uncertainty in ML estimates
- Generating distributions instead of point estimates – sampling at different quantiles can improve quality of estimates
- Can exercise judgement and switch between pessimistic & optimistic estimates with a single model
- No active uses in the actuarial domain yet, but there is potential

www.luxactuaries.com
navarun.jain@luxactuaries.com