Predicting Loan Level Mortgage Loss

An exploration of machine learning methods

Chris Walker

Table of contents

1 Introduction

This document describes the motivation, data, estimation, and validation of a credit loss model which compares non-parametric machine learning techniques to predict realized mortgage-level losses in basis points (Bps) of origination UPB. Moreover, this model is designed to estimate gross loss which represents risk to the financial system and not any one party (Freddie Mac, mortgage insurers, etc). The motivations for this project are twofold:

1. Study the relationship between common loan application details (credit score, debt-to-income ratio, loan-to-value ratio, etc) and mortgage losses.
2. Evaluate a range of machine learning methods which can be used to model complex regression problems, many of which contain non-linear relationships.

Obtaining good estimates of expected gross loss in a stress scenario is useful for healthy management of the financial system, particularly when losses are estimated early in the mortgage life cycle such as the time of application. It is worth noting that extreme mortgage losses are rare and are generally contained to stressed economic periods, including the great recession period. Thus, this model aims to predict loan-level loss for a mortgage given a stress scenario.

First, a few definitions:

Table 1: Definitions used in credit modeling

Term	Abbreviation	Definition
debt-to-income	DTI	Ratio of debt to a person's income
combined loan-to-value	CLTV	Ratio of loan balance to property value
credit score	FICO	Representation of credit history risk
basis-points	Bps	0.01% of a percent
unpaid balance	UPB	Loan balance to be paid
mortgage insurance	MI	Insurance paid to offset losses
delinquent	DQ	Failure to make payments

2 Data

2.1 Target Variable

Data for this project comes courtesy of Freddie Mac's loan level data repository. This data repository contains anonymous mortgage data in two primary components; origination data and over time data. Origination data contains information about a loan at its origin. This contains fields like the borrowers' UPB, FICO, LTV, and DTI when the loan was created. Alternatively, the over time data contains monthly summaries of mortgage activity as a borrower pays down their mortgage. This over time data contains information about losses. A mortgage loss occurs when a borrower fails to make payments and enters delinquent status. The financial system incurs losses when borrowers become delinquent for an extended period of time.

We compute loss as risk to the financial system. This means that it is possible to compute the amount of losses incurred by Freddie Mac (for example, losses minus insurance payments) but instead we compute total (or gross) losses incurred by every involved party. Our loss computation follows this general form:

$$\text{Gross Loss} = \text{Delinquent Interest} + \text{Current UPB} + \text{Expenses} - \text{Net Sale}$$

Delinquent interest is loan interest which accumulates when a borrower fails to make payments. Similarly, Current UPB denotes the loan balance at the time of delinquency. Once Freddie Mac repossesses the property they will attempt to sell the property. When the property finally sells much of the original loss is recouped (and sometimes a profit is made). However, in most cases there is still a gross loss balance which is positive representing loss to the financial system. These losses can be offset by MI assistance, but they are not included in this calculation.

$$\text{Gross Loss Bps} = 10000 \times \frac{\text{Gross Loss}}{\text{Origination UPB}}$$

We divide gross loss values by the original loan balance (origination UPB) and convert them to Bps. This provides us a value representing the percentage of a loan's original value which we might expect to be lost given a delinquency in a stressed economic scenario. Because we wanted to simulate losses in a stressed environment, we utilized loans originated right before the financial crisis (2007 acquisitions). However, not all loans in this population experienced a loss. Using the user guide provided by Freddie Mac, we restrict to loans which satisfy these conditions:

- Is a third party sale, short sale or charge off, REO disposition, or whole loan sale
- Does not have a populated defect settlement date
- Has all components of the loss calculation present (not missing)

The Freddie Mac data does include a net loss calculation as part of the data set. We compute our target variable from scratch because we want to measure risk to the financial system. However, the included net loss value is set to zero if one of the last two conditions listed above is not present. Thus, the training data for this model applies the following two conditions:

```
zero_bal_code in ('02', '03', '09', '15')
and net_loss != 0
```

Our target variable has the following distribution:

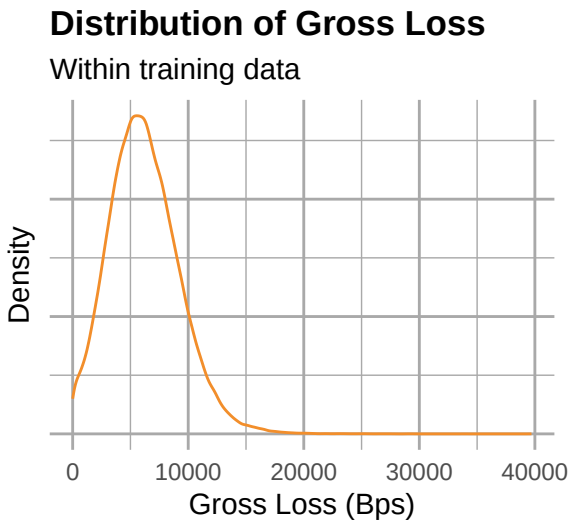

Table 1: Varable distribution

2.2 Independent Variables

This model makes use of loan level application characteristics to predict loss in basis points in origination UPB given a stressed economic scenario. While more sophisticated models can be estimated when more data is present (for example, early payment history data) we intentionally constrain this model to application level details to act as an early warning loss model. Fields which are present at the time of application include CLTV, DTI, FICO, occupancy status, property type, loan purpose, loan term, first time home buyer, and geographic region.

CLTV, DTI, and FICO are continuous values. This means that these values are a number which is intended to have a positive or negative relationship with gross loss. For example, as FICO scores increase, we would expect gross loss values to fall. All other predictors are categorical variables which means they indicate one of several options. For example, someone can live in one of four regions (but not more than one at once).

Here we perform a parameter stability exercise before leveraging more interesting machine learning methods. This method begins by randomly sampling the training data with replacement 100 times. From there, we estimate linear models and assess the stability of the coefficients. There are no additional transformations or splines applied to the continuous predictors, however the categorical variables are one hot encoded.

This means that the default values do not show up on the visual below. The coefficients measure the marginal increase or decrease over the default values for our categorical variables. For example, the default term is 30 years thus the two terms which show 15 and 20 years represent the marginal increase or decrease in expected loss over the 30 year category, holding all else equal.

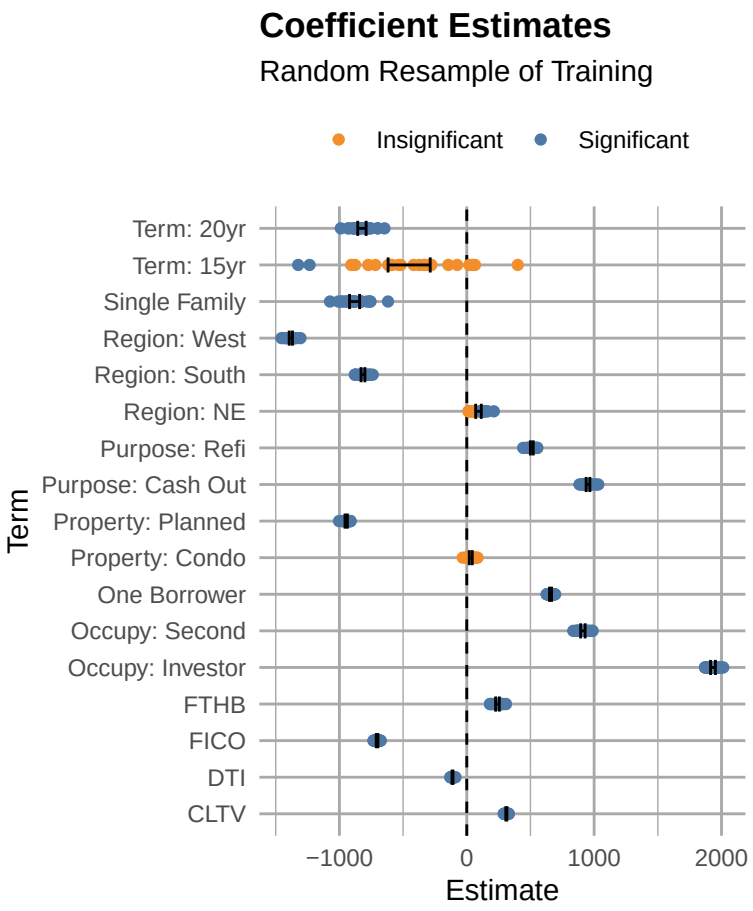

Table 2: Coefficient stability analysis

Most coefficients show relative stability with a tight confidence interval. Notable exceptions include 15 year mortgages, the northeast region, and condos. This means that the expected loss of these terms when compared to the defaults are not meaningfully different. Many of these estimations reveal sign flips and insignificant coefficients. The default categories for these terms are 30 years, the Midwest, and single family homes, respectively. This means we will treat these unstable categories as the defaults.

One other variable which might be unstable is DTI. This term tends to correspond with short term default over long term loss. While its confidence interval is small and it is always significant, the term has a negative coefficient which means we would expect losses to fall with increased DTI, which is counter intuitive.

All things considered, the primary terms for this model will be mortgage term, geographic region, loan purpose, property type, one unit, one borrower, occupancy status, first time home buyer, FICO, and CLTV. We will monitor the importance and direction of DTI across each learning method.

In the figure below we plot the distribution of each continuous predictor. Both DTI and FICO are approximately normal. The notable exception is CLTV which has large clusters at 80% and 100%. There is incentive for borrowers to put at least 20% down on their mortgage to avoid monthly mortgage insurance payments. Similarly, the cluster around 100% are borrowers who put very little down on their mortgage but still want to purchase a home.

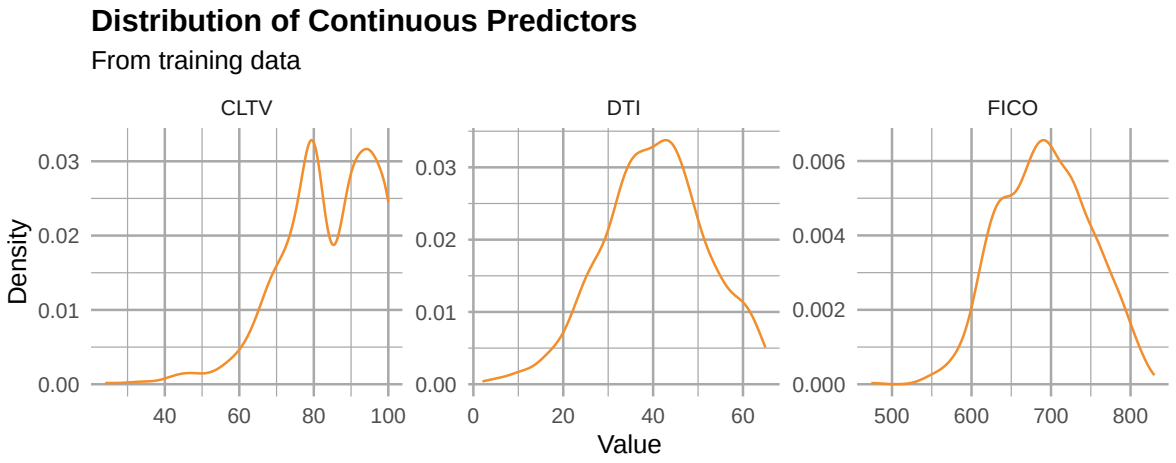

Table 3: Predictor distributions

Our complete data set contains 98489 records. We split the data into train and test. The train population is used for all model fitting procedures. We then use resampling approaches for interim steps (like hyperparameter tuning). Finally, we use the test population for final comparison between models.

Table 2: Train test splits

Population	Observations
Test	24623
Train	73866

3 Learning Methods

3.1 Linear Regression

3.1.1 Background

Our first modeling method is traditional ordinary-least-squares (OLS) regression, also known as linear regression. This method fits an additive model to the data by finding the coefficients or parameters

which minimize the sum of squared residuals (or error) between actual and predicted values. However, for continuous predictors with non-linear relationships, we can improve the fit of the model via piece wise splines. This allows the linear plane to have bends or knots which form a jagged shape that can better fit the data.

Lets begin by discussing a brief theoretical background for linear regression or OLS. Arguably the most fundamental and widely used modeling technique, OLS generates a numeric value (called a coefficient) for each variable in the model. It also generates a constant value (also called the intercept) which is used to correct the overall level of the model predictions. Linear regression makes predictions using the following structure:

$$\hat{y}_i = \beta_0 + \sum_{j=1}^{p} X_{ij}\beta_j$$

Here we want to get the prediction for observation i denoted as $\hat{y}_i$. The prediction is the intercept (denoted as β_0) plus the sum of p terms. In this case p refers to the number of predictors. We multiply each predictor for observation i (denoted as X_{ij}) by the beta for that term (denoted as β_p).

OLS attempts to minimize the sum of squared residuals. A residual is simply the difference between the actual value for this observation and the predicted value which we can show as $\hat{y}_i - y_i$. The sum of squared residuals is:

$$\text{Sum of Squared Residuals} = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

The values for β_j can take the form of any real number. Generalized models make use of a search process called gradient descent to determine the best values for each parameter. However, for OLS we can actually solve for these values making linear models very quick to estimate! It takes this form:

$$\beta = (X^T X)^{-1} X^T y$$

Note that here we refer to the model estimates as β. In truth these should be denoted as $\hat{\beta}$ because the coefficients themselves are estimates but we adopt this convention for this report. With our model in hand the last step is to estimate p-values for each term, including the intercept. P-values come from estimating standard errors for each term. We can then use these values to conduct t-tests for each term and assess the statistical significance of each β. Ideally, p-values are very small because small p-values indicate that there is a good chance that the true coefficients (as opposed to the estimated coefficients) are something other than zero and are meaningful.

3.1.2 Estimation

To construct our splines we will leverage the `earth` package in R which relates to multivariate adaptive regression splines (MARS). Running MARS over our train population results in one knot recommendation per continuous predictor. They turn out to be 720 for FICO, 80 for CLTV, and 37 for DTI. We reject the DTI knot because of DTIs already small (and somewhat counter intuitive) coefficient. Our OLS model has the following specifications:

Table 3: Linear regression specifications

Term	Estimate	P.Value
Intercept	7021	<0.001
FTHB	265	<0.001
Single Family	-901	<0.001
DTI	-105	<0.001
One Borrower	659	<0.001
Occupy: Investor	1887	<0.001
Occupy: Second	869	<0.001
Property: Planned	-960	<0.001
Purpose: Cash Out	881	<0.001
Purpose: Refi	464	<0.001
Term: 20yr	-819	<0.001
Region: West	-1404	<0.001
Region: South	-836	<0.001
FICO < 720	-845	<0.001
FICO > 720	-412	<0.001
CLTV < 80	423	<0.001
CLTV > 80	171	<0.001

3.2 XGBoost

3.2.1 Background

Extreme Gradient Boost (XGBoost or XGB) comes from a family of gradient boosted models. Specifically, XGBoost uses sequentially estimated decision trees which attempt to correct the residuals (or errors) from all prior trees. XGBoost begins by making a simplistic estimate and takes the average response variable for all train observations. The model then predicts this value for all training observations and computes the first set of residuals. The model then fits a full decision tree trying to predict these residuals using available predictors. If the model has a good understanding of residuals, then we can understand sources of variance in the response!

XGBoost uses something called Similarity Score to determine which variable and at which point along the variable (if it is continuous) produces clusters of data that are the most similar. After all, tree based models aim to bucket similar data together. Sufficiently similar data will get a representative prediction. A similarity score is computed as:

$$\text{Similarity} = \frac{(\sum \hat{y} - y)^2}{\text{Number of Residuals} + \lambda}$$

To prevent overfitting, XGBoost controls the impact of each tree with a learning rate and a shrinkage parameter. This reduces the impact of any single tree in the series and works to prevent overfitting or rote memorization of the training data. The trees' predictions are then combined, resulting in a robust ensemble that captures intricate relationships between features and the target variable. This learning method is powerful because:

- There is no need for splines or variable transformations
- It is robust to outliers and extreme values
- The algorithm is widely used and optimized for performance (e.g., multithreading)

However, it has some drawbacks. For example, it has a tendency to overfit despite controls put in place to prevent overfitting. It also does a poor job with extrapolation like other tree methods. This is because XGBoost assigns a value to data based on buckets. If your data is in a bucket at the top or low end of a variable's range (e.g., FICO > 780) the tree will assign the same value regardless of how far the observation is from the decision point (FICO 780 in the example here). Finally, XGB still struggles with interpretability like other non-linear methods. While metrics (e.g., frequency is a metric discussed later) help to explain feature importance, the model itself contains too many parameters and decision points to be well understood like a linear regression.

3.2.2 Estimation

Despite the drawbacks of XGBoost, it remains one of the best *batteries included* machine learning methods. This is because it is nearly as easy to estimate as a linear regression and offers many benefits (e.g., reduced need for feature engineering) right out of the gate. However, all XGB models should be fine-tuned to get the best performance. We used hyperparameter tuning with `tidymodels` to tune three key parameters:

- `tree_depth`: The number of nodes deep a tree grows.
- `learn_rate`: The relative impact of each tree.
- `n_trees`: The number of total trees in the model.

All of these values have a direct relationship with overfitting. However, setting these values too small means that the model will do a poor job of understanding the variance in our response variable. Thus,

we tuned these values. The default values and ranges come from `tune` in `tidymodels` as to prevent arbitrary value selection. After tuning we selected values of 6, 0.1, and 20 for `tree_depth`, `learn_rate`, and `n_trees`, respectively.

Using the feature importance metrics from `xgboost` we can get a sense of the contribution from each predictor. Where frequency represents the share of nodes across all trees which use each predictor, gain represents the marginal performance contribution of each predictor. A higher value corresponds to greater importance for both values.

FICO is the most important predictor followed by CLTV and DTI. Those three predictors alone dictate over 50% of the nodes in the tree as denoted by frequency in the table below. All other predictors merely assist those core predictors in understanding loss.

Table 4: XGBoost feature importance

Feature	Frequency	Gain
fico	25.5%	0.105
cltv	22.3%	0.065
dti	14.1%	0.024
purpose_c	6.3%	0.024
property_pu	6.1%	0.146
one_borrower	5.5%	0.058
occupancy_i	3.9%	0.226
region_south	3.5%	0.097
term_frm240	3.2%	0.005
region_west	2.5%	0.221
occupancy_s	2.4%	0.007
purpose_n	2.3%	0.007
one_unit	2.1%	0.016
fthb	0.3%	0.000

3.3 Neural Network

3.3.1 Background

The final model type we explored is a deep learning model estimated in Python using `torch`. A deep learning model is simply a neural network of two or more hidden layers. Models of this type can be built to solve extremely complex learning problems including computer vision, language modeling, and more. However, they can still be applied to more traditional predictive applications like this loss model.

A deep learning model for regression is a series of interconnected processing units. Each unit receives data (inputs) representing features, multiplies them by weights (learned values similar to regression coefficients), and sums them up. This sum is then passed through an activation function. The activation function decides how much influence each unit's calculation has on the final output. Common choices in regression include the ReLU function, which allows positive values to pass through unchanged, and zeroes out negative ones. This adds non-linearity, enabling the model to capture complex relationships.

With many interconnected units across multiple layers, each performing this calculation with its own weights and activation function, the deep learning model builds a complex web of influences. By comparing the model's predictions with actual data and adjusting the weights based on errors (back propagation), the model learns to transform the initial inputs into accurate predictions for a target variable.

3.3.2 Estimation

Our deep learning model, like XGBoost, needs to be tuned to optimize performance. We tuned a total of three parameters each hoping to better the overall regression capability of the model. The parameters are:

- `num_hidden`: The number of hidden nodes in each of the three hidden layers
- `dropout`: The share of randomly zeroed inputs (to prevent overfitting)
- `active_fn`: The activation function used (ReLU vs LeakyReLU)

Similar to the XGB tune, we utilized default values from `tidymodels` to initialize our tuning. For `num_hidden` we tried 10, 20, and 30 whereas we tried 0% and 20% for the number of randomly dropped inputs. Both of these parameters are designed to control for overfitting of the model. Models with less dropout and more hidden layers will fit the training data well but likely overfit and perform poorly on the testing data.

We also tested ReLU vs LeakyReLU as our activation function. As discussed above, these functions introduce non-linearity into the model to allow for complex relationships. The weight (or coefficient) simply adjusts the angle between the flat and angled portions of the activation functions. ReLU is a very popular choice because it is computationally efficient and simple to implement. The LeakyReLU expands on this idea by adding a slope to the flat portion to prevent exploding or vanishing gradients. Exploding or vanishing gradients are two extremes of the same problem related to the chain rule and convergence failures. After a full gamut of testing, we found that ReLU with 30 hidden nodes and 0% dropout leads to the best performance.

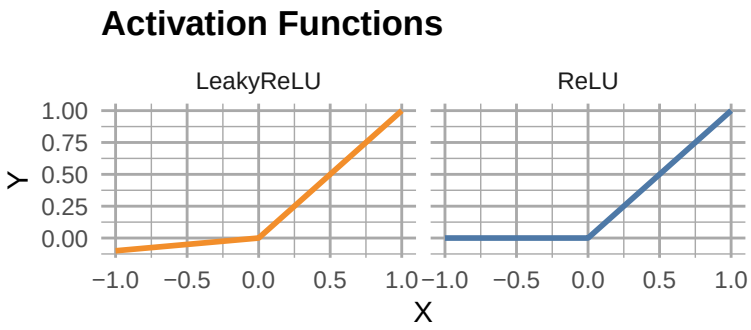

Table 4: Activation functions in deep learning

A deep learning network feeds forward by taking the design matrix, multiplying it with weights (hidden layers), and applying activation functions. In our case, our deep learning model contains three hidden layers and one output layer. It also uses the ReLU function. Assume that *X* is our design matrix and *W* are our weight matrices. We can denote the feed forward in pseudocode as:

```
# The design matrix
X = matrix()

# The activation function
ReLU = lambda x: max(0, x)

# Feed forward
for W in layers:
  X = ReLU(X @ W)

# Predictions as a column-vector:
print(X)
```

4 Performance Comparison

In this section we compare the relative model performance of each model using three metrics. These metrics are aimed at measuring the applicability of each model for different use cases. Specifically, Gini and TMR are designed to measure rank ordering performance whereas RMSE is designed to measure accuracy. Gini is directly related to the receiver-operator-curve or ROC. While ROC is more common in data science, Gini is common in credit risk applications and takes the following form:

$$\text{Gini} = 2 \times \text{ROC} - 1$$

Both Gini and ROC measure the model's ability to rank order a population. Thus they place greater emphasis on the model's ability to assign a higher value to riskier observations than a less risky observation regardless of the truthfulness of that actual value. This is in contrast to the square-root-mean-squared-error or RMSE which places more emphasis on accurate model prediction and takes the following form:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum(\hat{y}_i - y_i)^2}$$

Finially, tail-match-rate or TMR, like Gini, measures a model's ability to rank order. However, Gini measures rank ordering performance across the working range of our response variable. TMR measures the overlap between the highest 5% of actual loss values and 5% predicted values by asking the question "what percentage of the 5% highest loss values are within the 5% highest predicted values?" It takes the following form:

$$f(z, y) = \frac{1}{n}\sum I(\hat{y}_i \geq \text{95th percentile of } \hat{y} \text{ and } y_i \geq \text{95th percentile of } y)$$

Where *n* is:

$$n = \sum I(y_i \geq \text{95th percentile of } y)$$

In general we would like to maximize Gini and TMR to improve rank ordering while minimizing RMSE to reduce loss. However, in a credit decision setting we would prioritize rank ordering over accuracy. This is because most models are compared to some threshold which decides what is an acceptable amount of risk for an approved application. The specific threshold does not matter so long as those above the threshold are riskier than those below it as indicated by rank ordering performance.

If we leverage Gini as our dominant metric then XGBoost provides the best overall performance. This is meaningful because Gini measures rank ordering performance across the range of the response variable. However, if we use TMR or RMSE the deep learning model estimated using `torch` is the best performer for both train and test. Both models perform better than linear regression (OLS) despite the introduction of splines in our linear model.

Table 5: Model performance

Subset	Model	Gini	TMR	RMSE
Train	XGBoost	0.622	26.3%	2868
Train	Torch	0.599	26.9%	2759
Train	Regression	0.583	25.1%	2795
Test	XGBoost	0.598	26.5%	2872
Test	Torch	0.583	27.9%	2751

Subset	Model	Gini	TMR	RMSE
Test	Regression	0.576	25.6%	2783

While it may seem that we would have difficulty choosing between XGBoost and `torch`, the deep learning model was significantly more time and research intensive to develop. Furthermore, it was much more sensitive during our tuning phase while also taking much longer to converge!

This does not discredit the usefulness of deep learning models. However, it does suggest that relative to XGBoost, deep learning models are more challenging to develop particularly for narrow predictive applications. Likewise, XGBoost and its associated frameworks have made strides in developing useful model constraints. For example, monotonicity controls which allow modelers to specify the type of relationship a variable should have with the response (positive or negative). It is for these reasons that our **champion model is our tuned XGBoost**. We recommend this model type for a wide variety of modeling applications.

5 Resources

- Linear regression with tidymodels
- XGBoost with tidymodels
- Deep learning with torch
- Freddie Mac's loan level data repository