Online Appendix

to

Beating 'Best Estimates': Machine Learning Predictions, Managerial Errors, and the Warranty Provision

Martin Becker
Accounting Center Münster
University of Münster

Simon Schölzel*
Accounting Center Münster
University of Münster

February 2022

This online appendix provides additional texture on the preprocessing steps applied to the raw data and the employed machine learning methodology, alongside additional figures and tables. Note that a generalized version of our modeling script can be accessed via GitHub.

Contents

* Corresponding author: simon.schoelzel@wiwi.uni-muenster.de

OA. 1 Data Cleaning and Merging

In the following, we provide details on our data preprocessing pipeline. The exact same pipeline is applied to raw data from each of the three product lines.

Sample Period. We restrict our data to warranty claims that have been active during the 2010–2020 period and, thus, require a one-of provision estimate during that time. Importantly, we have complete historical data on expired warranty claims over the [–15; –3]-period relative to our first prediction date (fiscal year-end 2012). In addition, we also have full claim data up to 15 months after our last prediction date (fiscal year-end 2020) to ensure that all relevant claims have been filed, inspected and processed.

Prediction Date. We define for each product sale, i.e., warranty obligation, the prediction date as the first fiscal year-end date past the start of the warranty life cycle. At this particular date, an estimate of the warranty provision is generated by the human expert and the prediction machine, respectively.

Idenitication of Cost Variables. We derive the already incurred warranty cost by aggregating all claims settled until the respective prediction date, i.e., fiscal year-end. In contrast, we derive total warranty cost per product by aggregating all claims settled over the entire sample period. Further, we calculate the warranty cost which pertain to the remainder for the warranty life cycle beyond the fiscal year-end as the difference between the total and the already incurred warranty cost. This number is supposed to be reflected in the annual managerial provision estimate and, hence, serves as our response variable.

Geographical Predictors. We derive longitude und latitude predictors using the *geopy* Python package. In particular, we query the open OpenStreetMap API using various combinations of country, postal code, and city descriptions related to each product.

Merging. We mainly merge our different proprietary datasets via a unique product identifier. In addition, we partly merge on fiscal year-end dates and product types, e.g., to attach historical warrany cost predictors that have been computed via 12-months rolling window calulcations shifted over our entire sample period.

OA. 2 Warranty Provision Estimators

Managerial Estimators (f^{Acc}). In line with section 3.1 of our paper, we denote $f^{AccBase}$ as the managerial accounting estimator which only incorporates average historical cost information and $f^{AccFull}$ as the estimator which also incorporates managerial judgment in the form of rule-based adjustments and soft information. Since we are interested in the performance of machine vis-à-vis human predictions, we benchmark all subsequently trained machine learning estimators $\hat{f}^{ML}$ against the arguably richer $f^{AccFull}$ estimator.

Linear Regression (f^{LM}). We start with the family of linear models by training a regular ordinary least squares (OLS) regression to predict the projected warranty cost at the fiscal year-end. In addition to the managerial estimator $f^{AccFull}$, we use the linear regression model predictions $\hat{Y}^{LM}$ as a second benchmark for all subsequent models to assess the relative benefit of training more complex estimators.

Regularized Regression (f^{RR}). The elastic net model (Zou and Hastie [2005]) expands the OLS framework by introducing a regularization term. This term shrinks model coefficients of irrelevant predictors towards zero and thereby prevents potential overfitting. The elastic net exposes two main hyperparameters[1] to the researcher: the penalty λ determines the extent to which model coefficients are penalized in the OLS objectives, while the mixture parameter α determines the relative importance of two types of penalties, the Ridge (Hoerl and Kennard [2000]) and the least absolute shrinkage and selection operator (LASSO) penalty (Tibshirani [1996]). This setup enables the model to perform variable selection while being stable in the presence of correlated predictors.

Random Forest (f^{RF}). The random forest (Breiman [2001]) is a non-parametric ensemble model that does not presume a particular functional relationship between $\boldsymbol{X}$ and Y ex ante. In our setting, it is built by aggregating a large set of independent regression trees which are fitted on both, randomly sampled subsets of $\{\boldsymbol{X}, Y\}^{Tr}$ and randomly

[1] For a concise overview of all employed hyperparameters see **Error! Reference source not found.**. Models are implemented using the R programming language and *tidymodels* (https://www.tidymodels.org) as the underlying modeling framework. If we do not explicitly discuss a model's hyperparameter, we rely on the default value specified by the respective software package.

sampled predictors from $\boldsymbol{X}$ (Hastie et al. [2009]). Each tree recursively splits the predictor space into mutually exclusive subregions and models Y as the average value per subregion. Thereby, optimal split points are determined by reducing the variance of Y within each subregion. The final warranty provision estimate is then given by averaging over the independently trained trees. We search for the optimal random forest configuration by varying two important hyperparameters: *min_n* determines the minimum number of data points per subregion to continue the splitting procedure while m_{try} denotes the number of randomly selected predictors at each split point. We select a sufficiently large number of trees (i.e., 1,000) and use bootstrapping to create random subsamples of $\{\boldsymbol{X}, Y\}^{Tr}$. In particular, we sample from $\{\boldsymbol{X}, Y\}^{Tr}$ without replacement until we reach 60%, 80% and 100% of the original training set, respectively, to account for the varying computational complexities of our three product lines.

Gradient Boosting Machine (f^{GBM}). The gradient boosting machine (GBM) (Friedman [2001]) rests on the idea of constructing a stage-wise additive model consisting of a sequence of base models, most commonly individual decision tree stumps. After fitting such a base model, the gradient with respect to the model (i.e., residual) is computed and the subsequent base model is then trained to approximate the remaining residual. Ideally, each added base model contributes to reducing the error of the overall boosted model. Whereas the previously discussed random forest algorithm aggregates individual trees by averaging predictions, the GBM aggregates individual trees by adding them up. We rely on the efficient XGBoost algorithm (Chen and Guestrin [2016]) for model training. In addition to the random forest hyperparameters, we systematically vary the following parameters: *trees* denotes the number of sequentially added trees (i.e., boosting iterations), *tree_depth* denotes the maximum number of splits per tree, η denotes the learning rate (i.e., the magnitude of the contribution of each base model to the boosted model), and *sample_size* denotes the size of the subsample of $\{\boldsymbol{X}, Y\}^{Tr}$ used for model fitting at each iteration. We employ early stopping to prematurely end the algorithm if the model error does not decrease for at least 25 iterations.

Support Vector Regression (f^{SVR}). The support vector regression (SVR) is an adaption of the classical support vector machine (Vapnik [2000]) to the regression context (Drucker et al. [1996]). Thereby, a regression is fitted to those training samples for which the absolute residual is greater than a predefined tolerance margin ϵ. This procedure yields a ϵ-insensitive hyperplane which functional form depends on the more extreme data points, the so-called support vectors. In addition to the parameter ϵ, the loss function of the SVR stipulates a cost parameter C which penalizes large absolute residuals beyond ϵ. For large values of C, the regression fit gets more flexible as fewer support vectors are determined to anchor the hyperplane. Finally, we employ SVR with a radial basis kernel function to project our predictor set into a higher dimensional space and capture non-linearities between $\boldsymbol{X}$ and Y. This kernel exposes the hyperparameter σ which controls the influence of a single training sample on the fit. We optimize it in conjunction with ϵ and C. Since the computation time increases exponentially in the size of $\{\boldsymbol{X}, Y\}^{Tr}$, we use random sampling without replacement to reduce the training set size by 50% for our largest product line (i.e., product line 1).

Model Stacking (f^{MS}). Lastly, we apply the concept of model stacking (Wolpert [1992], Breiman [1996]). Similar to the idea of a group of decision-makers collectively forming an estimate of the warranty provision, our stacked model linearly combines the predictions of multiple base models (henceforth referred to as *singular* models). It is trained using the LASSO (Tibshirani [1996]), taking as input the predictions of the best performing linear regression, regularized regression, random forest, GBM, and SVR models. Since the LASSO performs variable selection via the penalty parameter λ, we can report the non-negative model weights to quantify the contribution of each model to the stacked model.[2] Model stacking rests on the idea that the stacked meta model optimally combines the individual predictions by trading off the strengths and weaknesses of the singular models in predicting the projected warranty cost, i.e., warranty provision.

[2] See **Error! Reference source not found.** for a summary of the optimal model stack weights for each singular model.

OA. 3 Model Training Scheme

Prior to model training, we apply some model-agnostic as well as model-specific preprocessing steps to our data in order to cater to the peculiarities of each model and training algorithm and facilitate the learning process. First, we pool rare levels in our categorical variables by creating an 'other' category. We enforce this step for every model to prevent the creation of (near-)zero variance predictor dummies which may impede the learning capabilities of some of our estimators. We treat the relative frequency *threshold* for pooling as a hyperparameter. For example, if it is set to 0.05, all categorical variable values with a relative frequency below 5% will be assigned the value 'other'. With regard to the model-specific preprocessing steps, we convert categorical variables into binary dummies for all estimators, except for the random forest model which is capable of handling categorical predictors as-is. Moreover, we center and scale all numeric predictors before training our regularized regression and SVR models, since the respective loss functions are not scale-invariant and depend on the magnitudes of the model coefficients.

We survey the space of possible model configurations using three different tuning algorithms. First, we employ regular grid search to iterate over a grid of hyperparameter candidates to determine the optimal model configuration. We use this approach in the context of our linear and regularized regression, random forest, and GBM models. Second, we employ model racing as proposed by Kuhn [2014] for our SVR estimators due to their heavy computational footprint. With model racing, we first train and validate model configurations on three randomly chosen time slices. Next, we discard all hyperparameter combinations for which the one-sided null of equal or better performance compared to the current best configuration can be rejected at the 5%-significance level using ANOVA. This leads to a large battery of configurations being discarded from further analyses which speeds up the training procedure. We repeat this process after each consecutive slice. Third, we use 25-fold bootstrapping on the validation sets to determine the optimal weights for our model stack. We summarize all hyperparameters and their respective search spaces in Table 1 as well as the optimal model stack weights in Table 2.

Table 1 – Hyperparameter Search Space

Hyperparameter	Search space	Levels	Total fits
Linear Regression			6
threshold	[0.00; 0.25]	6	
Regularized Regression			2,646
threshold	[0.00; 0.25]	6	
λ	$10^{[-10;\ 0]}$	21	
α	[0.00; 1.00]	21	
Random Forest			15,876
threshold	[0.00; 0.25]	6	
trees	1,000	1	
min_n	[20, 60]	21	
$m_{try}{}^{\dagger}$	[1, 21]	21	
sample_size	0.60, 0.80, 1.00	1	
Gradient Boosting Machine			48,000
threshold	[0.05; 0.20]	4	
trees	[50; 150]	5	
min_n	[2; 10]	5	
$m_{try}{}^{\dagger\dagger}$	[20; 100]	4	
tree_depth	[2; 5]	4	
η	{0.01, 0.05, 0.10, 0.20}	4	
sample_size	[0.50; 0.90]	5	
Support Vector Regression			4,356
threshold	[0.00; 0.25]	6	
ϵ	[0.00; 0.25]	6	
C	$2^{[-5;\ 5]}$	11	
σ	$10^{[-10;\ 0]}$	11	
sample_size	0.50, 1.00, 1.00	1	
Model Stacking			16
λ	$10^{[-10;\ 5]}$	16	

This table describes our hyperparameter search space, grouped by model family. For each hyperparameter, it states the search range as well as the number of evenly-spaced values sampled from this range. The column 'total fits' lists the total number of fitted model configurations per model family. Note that due to the large amount of different hyperparameters in the GBM model, we narrowed down the search space by running preliminary ceteris paribus experiments, i.e., varying a single hyperparameter while keeping all other constant, starting with the expectedly most important parameter.

†: Search space prior to dummy encoding

††: Search space subsequent to dummy encoding

Table 2 – Model Stack Coefficients

Estimator	Product line 1	Product line 2	Product line 3
Linear regression	0.2534		
Regularized regression			0.0566
Random forest	0.3513		0.0063
Gradient boosting machine	0.3775	0.5494	0.5537
Support vector regression	0.1814	0.7072	0.4409

This table presents the coefficients of the stacked LASSO model. It linearly combines the predictions of our singular prediction models (i.e., linear regression, regularized regression, random forest, GBM, and SVR). Empty cells indicate that the given estimator is omitted by the model stack. The higher the coefficient, the more weight the model stack assigns to the respective estimator. Weights are only stated for the models excluding the managerial adjustments ($\boldsymbol{X^{adj}}$).

OA. 4 Model Interpretation Techniques

In this section, we provide intuition for the model explanations that we employ in section 5 of our paper to facilitate the comparison of human and machine prediction errors.

Accumulated Local Effects Plots. Accumulated local effects (ALE) plots present a visual tool for understanding the isolated effect of a predictor X on the model's predictions $\hat{Y}$ (Apley and Zhu [2020]). Hence, they can inform us about the specific functional relationship between X and Y. The underlying algorithm can be summarized as follows:

(1) Split the data into equally sized intervals based on X (e.g., percentiles).
(2) For each data point, replace the value of predictor X by the interval's respective lower as well as upper bound (the other predictors remain unchanged).
(3) Generate predictions for Y using $\hat{f}^{ML}$ as well as the two mutated predictors and take the difference of the two predictions (i.e., $\hat{Y}_{upper} - \hat{Y}_{lower}$). This step keeps the conditional distribution of the unchanged predictors intact.
(4) Calculate the average difference in prediction over all data points in a given interval. This is referred to as the *local effect* of X on $\hat{Y}$.
(5) Summing up the local effects yields the *accumulated local effect* of X on $\hat{Y}$.
(6) This effect is centered by subtracting the mean of $\hat{Y}$ to enable straight forward interpretation, i.e., each point on the resulting ALE curve can be interpreted relative to the predicted mean.

This approach effectively elicits the main effect of X on $\hat{Y}$, even in sparse regions of X, while being robust to correlations among the predictors (Apley and Zhu [2020], Molnar [2021]). As a result, an ALE plot illustrates how the predictions of $\hat{f}^{ML}$ change over different intervals of X.

SHAP Values. Lundberg and Lee [2017] propose SHAP (SHapley Additive exPlanations) values as a framework for interpreting the predictions of complex machine learning models, based on seminal ideas in cooperative game theory. They are attractive due to their sound theoretical foundation and have been employed by related works as well

(e.g., Erel et al. [2021]). SHAP values can be calculated for an individual predictor $\boldsymbol{p}$ and data point $\boldsymbol{i}$ and quantify the contribution of this predictor to the difference between the model's prediction for this data point ($\hat{y}_i^{ML}$) and the average predicted value ($\sum_{i=1}^{n} \hat{y}_i^{ML}/n$). The main calculation steps are outlined below (Molnar [2021]):

(1) Generate K predictors masks $\boldsymbol{z} \in \{0,1\}^P$ with P being the total number of predictors. When multiplied with the predictor vector $\boldsymbol{x_i}$ of a given data point, some predictors are effectively masked out while the remaining predictors (the *coalition*) still contribute to the prediction.

(2) Construct two 'artificial' predictor vectors $\boldsymbol{x_{i,1}}$ and $\boldsymbol{x_{i,2}}$. $\boldsymbol{x_{i,1}}$ retains all original predictor values in $\boldsymbol{x_i}$ for which $\boldsymbol{z} = 1$. All remaining predictor values for which $\boldsymbol{z} = 0$ are replaced by predictor values randomly sampled from other data points in the dataset. $\boldsymbol{x_{i,2}}$ is identical to $\boldsymbol{x_{i,1}}$ except the original predictor value for the predictor of interest $\boldsymbol{p}$ is retained as well.

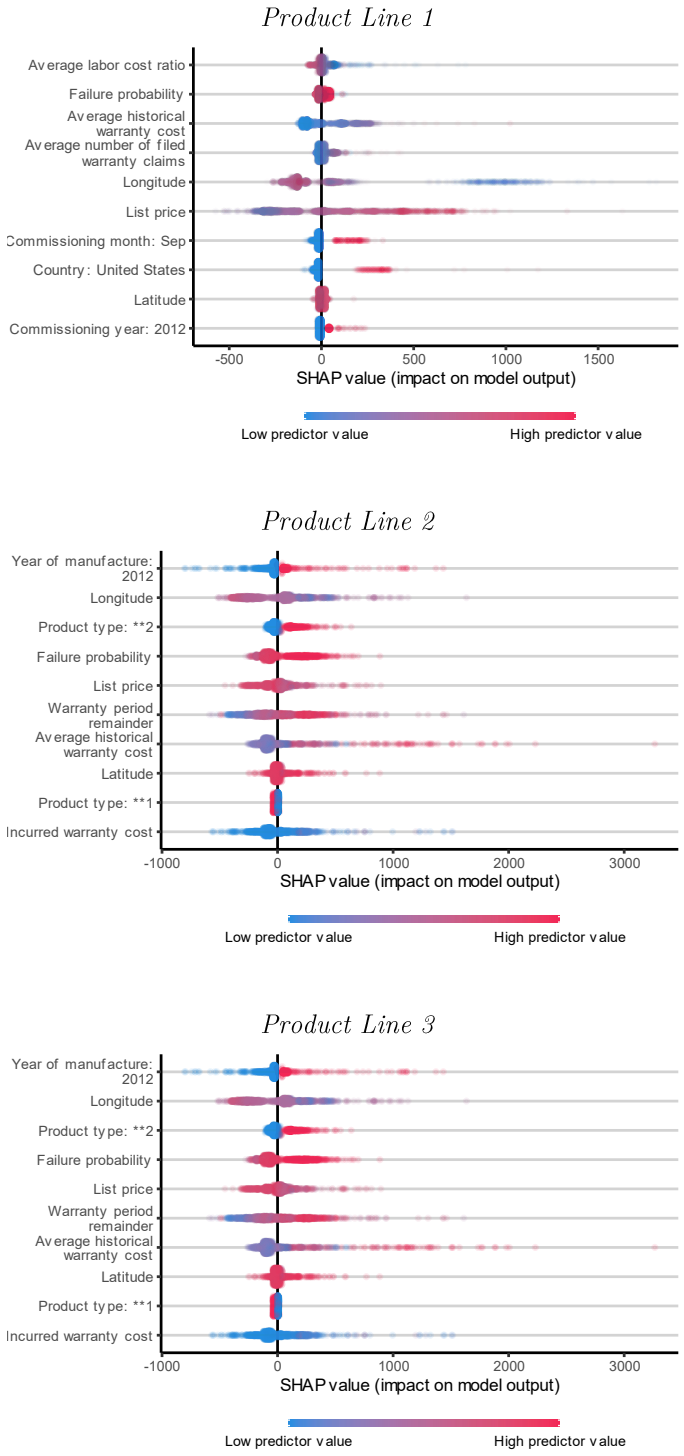
(3) Use the trained model $\hat{f}^{ML}$ to generate predictions for $\boldsymbol{x_{i,1}}$ and $\boldsymbol{x_{i,2}}$.

(4) Compute the difference of these two predictions (the *payout* to predictor $\boldsymbol{p}$).

(5) Repeat the process for all possible permutations of predictor masks. The SHAP value of predictor $\boldsymbol{p}$ for data point $\boldsymbol{i}$ is given by the average over all differences in predictions as computed under step four. In their initial paper, Lundberg and Lee [2017] propose a computational shortcut by leveraging a (weighted) OLS regression to compute predictor attributions for each predictor in P.

By averaging SHAP values per predictor $\boldsymbol{p}$ over all data points, we can obtain global predictor importance scores that enable us to sort predictors by their average importance. Since the computation time of the above procedure increases exponentially in the number of predictors and data points, we compute SHAP values using TreeSHAP (Lundberg et al. [2019]), an efficient implementation of SHAP values for tree-based models, such as random forest or GBM.

OA. 5 Disaggregatd Predictor Importance Rankings



This figure presents disaggregated importance rankings predicated on SHAP values (Lundberg and Lee [2017]). It lists the ten most important predictors (on the y-axis) for our trained GBM estimator ($\hat{f}^{GBM}$) across all three product lines. Predictors that contribute positively (negatively) on average are colored in red (blue). Importance is measured in terms of the mean SHAP value (on the x-axis), i.e., the contribution to the difference between the prediction and the mean prediction.

References

Apley, D. W., and J. Zhu. "Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models." *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 82 (4) (2020): 1059–1086.

Breiman, L. "Stacked Regressions." *Machine Learning* 24 (1) (1996): 49–64.

Breiman, L. "Random Forests." *Machine Learning* 45 (1) (2001): 5–32.

Chen, T., and C. Guestrin. "XGBoost. A Scalable Tree Boosting System." *Proceedings to the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016): 785–794.

Drucker, H; C. J. C. Burges; L. Kaufman; A. Smola; and V. N. Vapnik. "Support Vector Regression Machines." *Proceedings to Advances in Neural Information Processing Systems (NIPS 1996)* 9 (1996): 155–161.

Erel, I; L. H. Stern; C. Tan; and M. S. Weisbach. "Selecting Directors Using Machine Learning." *The Review of Financial Studies* 34 (7) (2021): 3226–3264.

Friedman, J. H. "Greedy Function Approximation. A Gradient Boosting Machine." *The Annals of Statistics* 29 (5) (2001): 1189–1232.

Hastie, T; R. Tibshirani; and J. Friedman. The Elements of Statistical Learning. Data Mining, Inference, and Prediction. 2nd ed. New York: Springer, 2009.

Hoerl, A. E., and R. W. Kennard. "Ridge Regression. Biased Estimation for Nonorthogonal Problems." *Technometrics* 42 (1) (2000): 80–86.

Kuhn, M. Futility Analysis in the Cross-Validation of Machine Learning Models. Unpublished paper (2014-05-28), Pfizer Global R&D, 2014. Available at https://arxiv.org/abs/1405.6974.

Kuhn, M., and H. Wickham. Tidymodels, 2020. Available at https://www.tidymodels.org.

Lundberg, S. M; G. G. Erion; and S.-I. Lee. Consistent Individualized Feature Attribution for Tree Ensembles. Unpublished paper (2019-03-07), 2019. Available at http://arxiv.org/pdf/1802.03888v3.

Lundberg, S. M., and S.-I. Lee. "A Unified Approach to Interpreting Model Predictions." *Proceedings to Advances in Neural Information Processing Systems (NIPS 2017)* 30 (2017): 4765–4774.

Molnar, C. Interpretable Machine Learning. A Guide for Making Black Box Models Explainable. Victoria: Leanpub, 2021.

Tibshirani, R. "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society. Series B, Statistical Methodology* 58 (1) (1996): 267–288.

Vapnik, V. N. The Nature of Statistical Learning Theory. 2nd ed. New York: Springer, 2000.

Wolpert, D. H. "Stacked Generalization." *Neural Networks* 5 (2) (1992): 241–259.

Zou, H., and T. Hastie. "Regularization and Variable Selection via the Elastic Net." *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 67 (2) (2005): 301–320.