Reducing E-Commerce Return Costs - A Prescriptive Profit-Conscious Ensembling Framework*

Justin Engelmann

School of Business and Economics,
Humboldt-Universität zu Berlin

March 18, 2019

Abstract

Item returns in e-commerce are costly both for the E-tailer and the environment. If we can predict when a customer will likely return his order before the order is made, we can target them for an intervention to try to dissuade the purchase. To account for asymmetric, item price dependent error-costs, we calculate the optimal threshold for intervening dynamically for each order. Furthermore, we use Ensemble Selection to improve our performance in terms of both AUC and profit compared to the best base model. Finally, we achieve further small gains through Profit-Conscious Ensemble Selection.

1 Introduction

Customers nowadays have come to expect being able to return their online orders for free. For etailers focussed on fashion, return rates can even exceed 50% (Asdecker, 2015; cited in: Urbanke et al., 2015). This a major cost driver and threat to profitability. Thus, reducing returns has become a top priority for businesses. However, due to customer expectations, no major etailer wants to risk upsetting customers and losing market share by charging for returns.

If it can be predicted that a customer will return his order *before* an order is made, however, then etailers could intervene to dissuade that purchase. Possible interventions

*This is a term paper for the course "Business Analytics and Data Science" at Humboldt-Universität zu Berlin.

are offering fewer payment options, especially not offering pay-after-delivery; telling the customer the item is out of stock; warning the customer that returns hurt the environment; outright refusing the transaction. (Urbanke et al., 2015) Erroneously intervening when a customer would have not returned an order is much more costly than erroneously *not* intervening with something that ultimately is returned. In the former case, we miss out on revenue and risk customer loyalty; in the latter, we only occur logistics costs and minor opportunity costs of the item being temporarily out of our inventory.

Thus, we present a smart prescriptive framework that accounts for asymmetric error costs to make profit-maximising decisions. For that we first create an effective predictive model and then calculate the optimal intervention threshold τ based on order-specific error costs to make a decision. Finally, a Profit-Concious-Ensemble-Selection Strategy is used to maximise our performance through a selective ensemble of heterogeneous base models.

2 Exploratory Data Analysis

We received a data set with 150.000 orders, with 100.000 being labelled indicating whether the order was returned or not. The data has 12 columns describing the order (e.g. order date), the user ordering (e.g. user ID), or the item (e.g. item price). 4 of the features are dates, 4 categorical, 3 IDs, and 1 continuous. About 10% of the values for delivery date and user date of birth (DOB) are missing.

We checked whether the labelled and unlabelled data sets are consistent, which they largely are, i.e. both cover the same time period, the distributions of item price are identical and most user IDs occur in both data sets. However, for the brand and item IDs, there is little to no overlap. A distribution plot clearly illustrates that those IDs have been transformed between the two data sets and that we can easily recover the proper IDs.

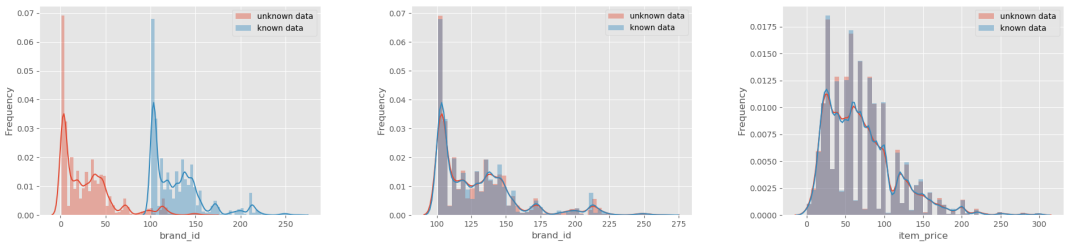

Figure 1: Distribution plots showing distribution plots for a) brand ID before adjustment, b) after adjustment, and c) for item price.

Another important observation is that for all orders where the delivery date is missing, the order has also not been returned. This could be interpreted as the customer cancelling

the order before it has been dispatched. If this is right, then this information is not very helpful for targeting customers before the order is finalised.[1]

3 Data Preparation

The transformed brand/item IDs were corrected. User DOB was transformed into age and missing values were mean imputed. Furthermore, 1.2% of DOBs were the 21st of Nov in 1900/1949. Those values were also mean imputed.

Item size included both numbers and letter codes. For simplicity, it was assumed that all items are women's clothing (96% of costumers are female) and sizes lower than 30 are non-European sizes. This was done for all levels that occur more than 100 times in the data; the rest were force-converted to integer and missing values mean imputed.

Order and delivery date were used to compute delivery time. Order date was used to create an ID (from 0 to 365) for each day in the data set, to capture any effects of seasonality and special sales like Black Friday.

For brands, items, colors, size, price, and states, average return rates per category (conditional expectations/CEs) were calculated.[2] CEs were only computed for categories that occur frequently to limit target leak. We also added CEs for item ID and color/size jointly.

We introduced a number of customer-specific variables, such as life-time-value (LTV), number of orders and return percentage/CE.[3] Furthermore, we added order-specific variables such as recency, number of items/colors/sizes ordered, how much the size differs from that user's average size, and whether the item was bought on discount.

For all variables, binning with similar amount of cases per bin was tried. Results were inconclusive, and best performance was achieved with binned and unbinned variables together. Variables were rounded to limit overfitting by tree-based classifiers. Some further dummy variables were introduced for imputed variables (to recover residual information).

4 Model Tuning and Selection

Prototyping and feature engineering was primarily done with Logistic Regression due to its speed. The following algorithms have been used: Logistic Regression (Logit), k-Nearest-Neighbors (KNN), Random-Forest-Classifier (RFC), Adaptive-Boosting-Classifier

[1] In this case, those orders should be more akin to returns than to non-returns. In the future, one could test labelling those orders as returns when training and then setting the predictions to 0 for all orders without delivery date.

[2] Weight-of-Evidence was also tested but did not improve model performance, even for Logistic Regression.

[3] Only added for frequent customers with more than 3 orders. Great care was taken to exclude each row's label for the calculation of that row's features. We also introduced some noise (inverse proportional to number of orders) to the CE and rounded them to further reduce target leak.

(AdaBoost, ABC), Extreme-Gradient-Boosting (XGBoost, XGB), Artificial-Neural-Network (ANN) and a Deep ANN (DNN). Support-Vector-Machine (SVM) was also tested but ultimately not used due to its poor performance and because SVMs do not organically offer probabilities, which we need in this application. Classification-And-Regression-Tree (CART,Tree), too, was tested and performed relatively well, but was excluded from further analysis in favour of the more advanced tree-based classifiers.

4.1 Tuning

4.1.1 Hyper-parameter Tuning

Models were tuned, with the hyper-parameter to tune chosen based on literature and scikit-learn/xgboost/keras documentation, using crossvalidated gridsearch and randomised crossvalidated gridsearch with the exception of ANN and DNN, which were tuned manually.

4.1.2 Classifier Probability Calibration

Since the goal is to not only predict costumer returns but to make a decision on whether to intervene, consistent probabilities are needed. This is because error-costs are asymmetric and dependent on item price (thus the optimal threshold varies). We want our probabilities to be consistent, that is: for all cases, where we predicted class membership probability of roughly X%, we want about X% of those cases to actually be in that class. To this end, classifier predictions were analysed with a calibration curve, which plots binned predicted probability against true ratio of cases in that bin. An ideal classifier follows a 45 degree line.

ABC has very inconsistent predictions. All predictions of ABC are very close to 0.5. (Empty bins are not plotted.) This means that ABC can perform well in terms of AUC (which is a ranking measure) and even accuracy, but its output should not be interpreted as probabilities. KNN and RFC also did not perform quite as well as the other algorithms.

To remedy this, ABC, KNN and RFC were calibrated using Platt's Scaling which uses a sigmoid function to scale classifier output. This was done using scikit-learn's CalibratedClassifierCV which takes the training data, splits it k-fold (we used k=3), builds k classifiers, evaluates them on their respective test folds and learns how to calibrate that model accordingly. For predicting on unknown data, all k models' predictions are averaged.[4]

As we can see, CalibratedClassifierCV improved our calibration curves for ABC substantially, an arguably also improved the other models slightly. The scaling is very computationally expensive because it requires building k-times as many models.

[4] A non-parametric isotonic regression version of Platt's Scaling was also tested but did not yield better results.

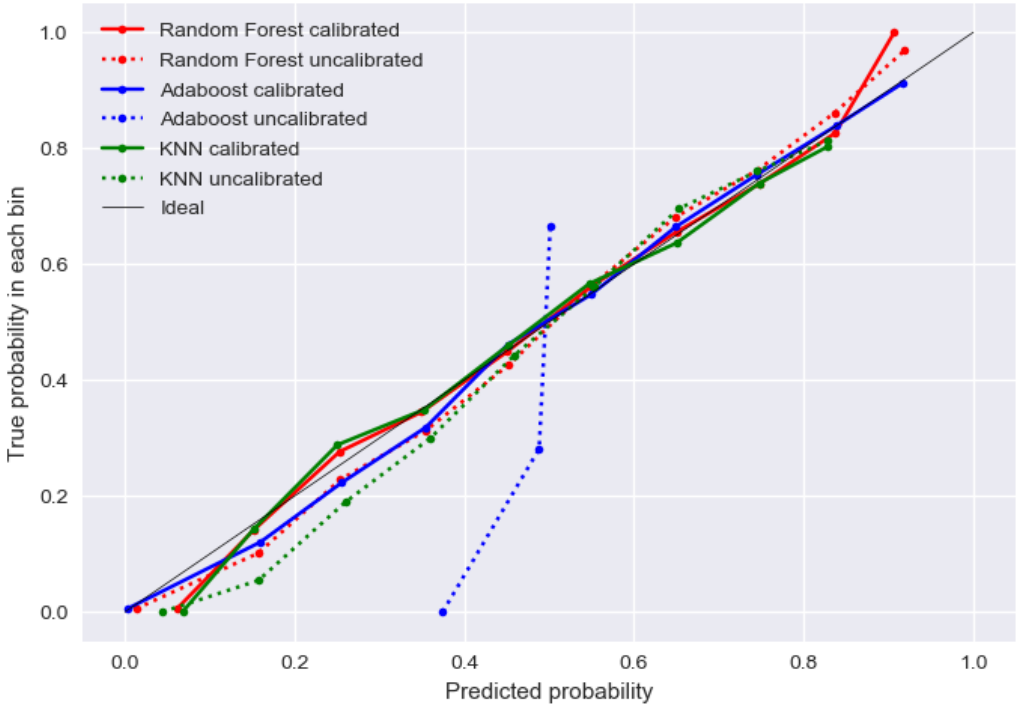

Figure 2: Calibration (reliability) curve for selected classifiers. The others performed quite well out of the box. The 45 degree line shows an ideal classifier.

4.2 Selection

The best single model performance in terms of Area-Under-Receiving-Operator-Curve (AUC) was determined based on 5-fold crossvaldiation (CV). XGB performed best, followed by RFC and ANN/DNN. Logit and KNN performed the worst but still easily achieved AUCs > 0.70. Furthermore, ANN/DNN's CV-performance generalised better to the validation set than that of XGB. This might be because ANN/DNN used early stopping, which might have also improved XGB's performance if used.

5 Model Evaluation

This section is about *how* models were assessed. Generally, k-fold CV was used. Kindly refer to Section 7 for results.

5.1 Kaggle

For the Kaggle competition, models were evaluated in terms of their AUC. The AUC is a ranking measure and can be roughly explained by saying: the AUC-score is the probability that a randomly selected case of class 1 will have a higher predicted probability of being in class 1 than a randomly selected case of class 0.[5]

5.2 Cost

For the final assessment, we evaluated models in terms of their average cost according to the cost matrix. However, since error costs are proportional to item price, the optimal threshold τ at which to issue a warning varies, too. Thus, we implemented a function which first calculates τ, compares it to the predicted probability to make a decision, and then scored our model based on the error costs.

6 Ensemble Selection

As our special modelling challenge, we implemented a selective averaging ensemble of heterogeneous base models (BMs). The heterogeneity was achieved through using different algorithms and hyperparameter settings. With a selective ensemble, there is a tension between strength and diversity: on the one hand, we want BMs to make good predictions, on the other hand, we want them to be diverse. The better our BMs perform, the better the performance we can get by averaging them. But if all of our BMs behave virtually

[5]This seems only approximately correct, since naively guessing the same value for all cases gets us an AUC of 0.5. However, the probability that any case has a higher predicted probability than any other case is exactly 0 (since they all have the same predicted probability).

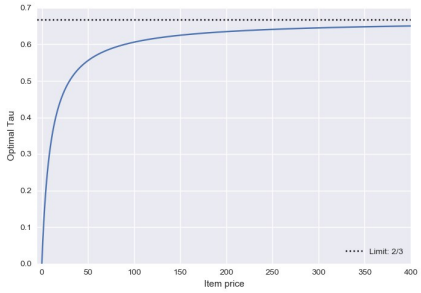

Figure 3: Shows optimal τ versus item price. It starts at 0 and asymptotically approaches $\frac{2}{3}$ as the size of the fixed delivery costs becomes small in relation to item price. While this approach happens quickly, recall that the distribution plot for item price shows that the majority of items cost under EUR100, so that the region with $\tau < 0.6$ is very important.

Table 1: **Model Zoo:** Algorithms and hyper-parameter settings.

Algorithm	Comments	Hyperparameters	Models
Logit	Variables selected with forward stepwise variable selection, stopping when AUC over multiple train_test_splits stops improving substantially.	Regularization factor C: 0.001-100 Penalty: L1/L2	12
ANN	Using Early Stopping with restoration of best model weights. Using ADAM as optimizer.	Activation function: relu/sigmoid Neurons per layer: 16, 32, 64 Batchsize: 32, 1024	9
DNN	Using Early Stopping with restoration of best model weights. Using ADAM as optimizer.	Activation function: relu/sigmoid Neurons per layer: 12, 32 Number of hidden layers: 2, 3	5
KNN	Using sklearn's implementation of Platt's Scaling. Only two models due to relatively poor performance and slow computation.	Number of neighbors k: 50, 400	2
RFC	Using sklearn's implementation of Platt's Scaling.	Number of trees: 100, 400, 600, 800 Covariates for splitting per tree: 0.5, 1, 2, 3 * sqrt(n_features) Minimum samples per split: 10, 20, 30, 40, 50	17
ABC	Using sklearn's implementation of Platt's Scaling.	Number of trees per ensemble: 10, 20, 30, 100 Depth per tree: 1, 2, 3	9
XGB	Covariates per tree set to 70%	Number of trees: 250, 400, 600, 800 Depth per tree: 2, 3, 4, 5 Covariates for splitting per level: 40, 60, 80, 100%	15
Total			**69**

identically, then the ensemble's predictions will be very similar to each BM's predictions and thus not improve much over them. Therefore, we chose not to use (randomised) grid-searched hyper-parameter settings but instead used a wide range of settings and let the Ensemble Selection (ES) algorithms select the most suitable BMs.

6.1 Model Zoo

Please refer to table 1 for the algorithms and hyper-parameter settings used to construct the model zoo. Note that with 6-fold CV and Platt's Scaling, this was very computationally expensive. Thus, the model library is more limited that we would have liked. Not all combinations of hyper-parameter settings were used.

6.2 Ensemble Selection Algorithm

For our ES, we proceed as follows: We split our labelled data into 6 equally-sized cross-validation folds. For each fold, we train all our models on the training set and obtain the

models' predictions for the test set. Thus, we end up with six vectors of predictions (one per fold) for each model. We also save the true labels and item price for each fold's test set.

Next, *for each fold*, we determine one winning ensemble based on the objective function. First, we evaluate each model's predictions and find the best base model (BBM), which is the first member of our ensemble.[6] Then, we try adding every model to the ensemble individually and evaluate it. We add the model which performs our ensemble's performance the most. Lastly, we iteratively repeat this procedure until no model improves the ensemble's performance.[7]

Thus, we obtain one winning ensemble per fold. We also create a further ensemble by concatenating each winning ensemble, effectively creating a weighted average. Finally, we crossvalidate each ensemble by computing its score on each fold. Our final ensemble is the ensemble with the best average performance across all folds.

6.3 Profit-Agnostic-Ensemble-Selection (PAES)

For PAES, we use a statistical objective function for the ES algorithm. In this case, we use AUC and select the BBM and ensembles based on their AUC-score.

6.4 Profit-Conscious-Ensemble-Selection (PCES)

PCES is an adaptation of PAES which has been recently proposed by Lessmann et al. (2019). It uses the ES algorithm but uses a problem-specific (as opposed to statistical) objective function. In our case, we use the cost matrix specified in the assignment description. This approach has the advantage that we leverage the solid[8] statistical foundations of the algorithms we use for model building, while also optimising our actual objective function. For instance, with error costs are proportional to item cost, we would rather make fewer errors for expensive items than for cheap items. Statistical cost functions like AUC do not take this into account while PCES does.

One disadvantage is that computing our problem-specific cost function is typically slower than computing e.g. AUC. However, both PAES and PCES can use the same model zoo, and constructing that zoo and training all the models takes orders of magnitudes more time than the ES algorithm does. So if PCES offers any noticeable improvement over PAES at all, it is worthwhile to use.

[6] We also experimented with adding a bias for the BBM by entering it multiple times into the starting ensemble so each further model has a smaller impact on the ensemble. This leads to a much greater diversity of models entering the ensemble, but to worse generalisation due to overfitting. Starting the ensemble with all models had a similar effect.

[7] Note that models can enter the ensemble multiple times. Thus, the ES algorithm results in a weighted average of models.

[8] For some algorithms: solid for Machine Learning standards.

7 Conclusion & Results

All of the tested algorithms were able to easily cross the threshold of 0.679 AUC, highlighting that feature engineering often trumps a finely tuned model.

PAES/PCES improved performance over the best single model in terms of cost and AUC both in local CV and on Kaggle. The AUC performance is surprising, since the previous best had been by a XGB model with hyper-parameters set by a comprehensive random gridsearch CV, whereas the model zoo had no gridsearched parameters.

In terms of costs, never intervening costs an estimated EUR12.82 on average per order, whereas the PCES ensemble achieves an average loss of EUR9.16 - an improvement of 29%! It also performed 6% better than a standard logit model.

The PAES ensemble achieves an average loss of EUR9.18 which is virtually identical to PCES. PCES/ PAES outperform the best base model (a RFC) by about only 1%. This might indicate a lack of strong, yet diverse models in the model zoo. The final PCES ensemble was two copies of one RFC, another RFC and a XGB. A few measures to increase diversity of the final ensemble were tested (as discussed in a prior footnote), but failed to improve CV performance.

Overall, we have shown that our prescriptive targeting framework can significantly reduce an etailer's return costs - in this case by almost 30%. However, while our selective ensemble (PAES/PCES) has improved our results over the best single model, the modest 1% gain is not enough to justify the much higher complexity and computational requirements.

References

[1] Stefan Lessmann, Kristof Coussement, Koen W. De Bock, Johannes Haupt, *Targeting customers for profit:An ensemble learning framework to support marketing decision making,* IRTG 1792 Discussion Paper 2018-012.

[2] Björn Asdecker, David Karl, Eric Sucky, *Examining Drivers of Consumer Returns in E-Tailing using Real Shop Data,* Proceedings of the 50th Hawaii International Conference on System Sciences, 2017.

[3] Patrick Urbanke, Johann Kranz, Lutz Kolbe, *Predicting Product Returns in E-Commerce:The Contribution of Mahalanobis Feature Extraction,* IZA Discussion Paper No. 1991, 2006.

[4] paazl.com, *Can you predict e-commerce returns? An analysis of 651.658 online orders and 412.584 returns,* https://www.paazl.com/blog/can-you-predict-e-commerce-returns-an-analysis-of-651-658-online-orders-and-412-584-returns/

[5] scikit-learn documentation, *Probability calibration,* https://scikit-learn.org/stable/modules/calibration.html

[6] Stefan Lessmann, Johannes Haupt, Alisa Kolesnikova, *Business Analytics and Data Science - Lecture and Tutorial slides, notes etc.*