BADS WS18/19 – Final Assignment

Alex Truesdale – 598854

2019.03.14

Introduction

In the section of the e-commerce world where physical products are sold, one of the greatest operational costs is these products being returned. With the growing availability of in-depth customer data and the increased sophistication / deployability of machine learning algorithms and statistical methods, firms can attempt to preempt returns and, in turn, improve their bottom line. This paper documents the case of clothing return predictions for an e-commerce company. There are 100,000 rows of labeled data (historical data – return status is known) upon which machine learning algorithms are trained to predict unencountered cases. 50,000 rows of unlabeled data (return status unknown) are provided, for which predictions on likelihood-of-return are produced. These predictions are then used in combination with a cost matrix to determine binary values from which business decisions can be determined – in this case, whether or not to show a warning message at the item checkout page to discourage the purchase. The report on the methodology and results of this work is broken down into the following sections:

1. Exploratory Data Analysis
2. Data Preparation
3. Model Tuning and Selection
4. Model Evaluation
5. Special Modeling Challenge (Ensembling)
6. Binary Classification & Conclusion

Exploratory Data Analysis

Before knowing how to work with a given data set, the structure of the data, its quirks, and any idiosyncrasies need to first be examined. The EDA process involves checking data types, frequency tables, and data visualisations to build such familiarity.

Raw data for this business case includes the following features:

```
order_item_id, order_date, delivery_date, item_id, item_size, item_color, brand_id
item_price, user_id, user_title, user_dob, user_state, user_reg_date, return
```

Frequency tables have proven quite useful in examining these given features, highlighting errors in data encoding (e.g. misspells in *item_color* and null values in date columns *delivery_date* & *user_dob*). The most important finding in this exploratory analysis, however, is the difference between often-occurring *item_id* and *brand_id* values in the known and unknown data. There is an immediately recognisable relationship between entries from the respective sets that suggests the IDs are not unique from one another but have been artificially transformed. Following this discovery, the IDs can be re-transformed, allowing features constructed upon *brand_id* and *item_id* to be much more useful.

This relationship is represented below by the following frequently appearing item and brand IDs:

- *Unknown[item_id]* (adjusted) = *known[item_id]* * 2 + 2
  - **known**: 4670, 2832, 66
  - **unknown**: 2334, 1415, 32
- *Unknown[brand_id]* (adjusted) = *known[brand_id]* + 100
  - **known**: 103, 105, 101, 137, 111
  - **unknown**: 3, 5, 1, 37, 11

In addition to frequency tables, taking the return ratio (count of returns over count of rows) for the levels of features provides quick insights into the overall target variance for a given feature. For *user_state*, as an example, no state is significantly above or below a return rate of .5. This feature therefore provides little predictive information and can be dropped.

In constructing these aggregate tables, it also becomes visible how diverse and un-unified some features are. In *item_size*, values range from strings (xs, s, m, l, xl, xxl, xxxl) to numeric measures on a variety of different scales (0-10, 30-60, 200+, etc.). Such observations in EDA inform subsequent decisions made in the data preparation stage, where features are engineered, repaired, or removed entirely.

Data preparation

Data preparation makes for ~80% of the work in performing machine learning analyses such as this, as the final predictive power of a classifier is highly dependent on quality inputs. Final features (28) fed to machine learning algorithms are as follows:

```
item_size, item_price, ce_brand, ce_item, ce_user, ce_price, is_discounted,
orders_per_day, multi_colour_per_day, colour_item_ce, count_item_user, customer_value,
count_user, new_user, size_item_ce, type_1, type_2, type_3, type_4, type_5, type_6,
type_7, was_delivered, latest_order, ordered_item_recently, membership_age_days,
time_to_delivery_days, assumed_age
```

The only feature that isn't modified or engineered is *item_price*. The others, with varying effectiveness (based on feature imporance reports from XGBoost and random forest algorithms), are generated out of the original features listed prior. The most useful of these engineered features are the *ce* values, standing for conditional expectation, which are calculated on their respective base features (brand, item, user, price, etc.). These features serve to provide insights with respect to the question: *using historical knowledge of orders grouped by unique values, can more granular future return behaviour be predicted?*

These features prove to be rather informative in production; however, there are two important limitations to examine:

1. Conditional expectation only provides information on already-seen users, items, etc. This does not, therefore, generalise to entirely new observations. This is not a problem, necessarily, but it does underscore the need for predictive features beyond those calculated solely from historical data.

2. This can easily lead to target leak and overfitting the data in training models if strong conclusions are drawn from too few obervations. For example, say a unique item ID appears twice in the data set and is both times returned. This is not enough information to then assume that items of this *item_id* will always be returned in the future. Supplying a threshold for count of observations over which these *ce* values are accepted builds more robustness into the feature (under-threshold values simply receive the *ce* value of .5 to minimise prediction effect).

Much less computationally complicated is identifying items that were never delivered. Doing so immediately and effectively classifies these cases as 0 (not returned) because there is nothing to return. Other date delta features that rank highly in importance identify cases in which a single user is purchasing multiple items on a single day, the same item in different colours, or purchasing multiple items within a short time window.

There is some colinearity amongst features (e.g. time_to_delivery_days & was_delivered both capture rows with undelivered items); however, this does not appear to negatively affect model preformance.

item_size values have been transformed to be represented as *type* features (types 1-7). These features provide little predictive value except for *type_7*, which denotes unsized items, which are perhaps less likely to be returned.

Note: many of the engineered features require multi-step aggregation, which the Python package Pandas is quite capable of. It does suffer, however, in indexing data at-scale in order to group by or assign new values. Writing functions utilising dictionaries as intermediaries between Pandas methods resulted in speed improvements of up to and exceeding 1000%.

Model Tuning and Selection

Tree-based algorithms (random forests, eXtreme Gradient Boosted trees) have been selected as the focal classifiers for this task. The data is not limited by target imbalance, having been artificially balanced to a 1:1 returned to not-returned ratio, nor are there any significant biases that put these algorithms at too great a risk of overfitting the train data. Given the significant popularity of these models in literature and informal online sources, they have been chosen as the starting point for model testing. The immediate excellence in testing performance and Kaggle leaderboard ranking has reinforced the models' suitability for the classification task, and, as such, they remain the primary algorithms in use.

Tuning these models' hyperparameters is then critical in managing the balancing act of fitting the data well without overfitting it, especially as XGBoost and random forests are both are prone to overfitting, if poorly tuned. To identify optimal hyperparamter values, the Python package scikit-learn's RandomizedSearchCV method is employed. With this, a range of randomised parameter combinations are tested in order to find their optimal levels. This is performed twice: the first iteration covers a grid with widely ranging parameter values in order to narrow in on a general space of successful hyperparameters. The second iteration focuses on a more specific grid of values within this space identified in the first search. These hyperparameters are tuned by cross validated AUC optimisation.

Ultimately, an ensembling of XGBoost and random forest models is carried out to increase predictive accuracy. To this end, a selection of 5-10 semi-similar versions of each model with parameters near to the determined optimal parameters has been made (explained further in the *Special Modeling Challenge (Ensembling)* section).

Model Evaluation

The primary method of model evaluation is the AUROC curve score (Area Under Receiver Operating Characteristic), shortened for simplicity to AUC. For classification problems, this metric well-encapsulates a classifier's ability to distinguish one class from another (in this case between customers who do or do not return their purchased item). All optimisation from hyperparamter tuning to ensemble selection is in the service of increasing this AUC score.

Simply maximising AUC is not a bulletproof strategy, however, as target leak and overfitting data also produce higher AUC values in model training / testing. When testing models, comparing test AUC scores to Kaggle AUC prediction scores indicates the degree to which overfitting and target leak are occuring. The following methods are then utilised to reduce this error:

1) Further feature engineering with more strict cutoffs on historical-data-heavy features. This is to avoid focusing too much on non-substantial past behaviour that might not generalise well to future prediction.

2) Stringent cross validation and model ensembling, both of which mediate one-off biases from 'lucky' training folds. By adding model diversity and averaging prediction outcomes, extreme model prediction behaviour in either direction is curtailed.

Taking Cost into Account

The ultimate goal of this case is to determine, in real time, which customers are going to return the item that they purchase and discourage such purchases. The models, however, produce probabilities ranging from 0 to 1 for how *likely* a return is for a given customer. These predictions now have to be reframed as yes or no business decisions (warn or do not warn customer at checkout).

For this, an error cost matrix is used to quantify the penalties of misclassifying a customer and choosing the incorrect course of action. In single classification, two types of error are possible: false positives and false negatives. These errors do not incur equal loss of revenue in this case and can be represented by a ratio that defines the point at which error types are balanced to minimise overall cost. This cost-minimal threshold is derived from item price and therefore assigns varying cost risk to items of different value where high-value items are approached with less conservatism and vice versa. Allowing a customer to purchase a high-cost item nets the firm a greater revenue, should the item not be returned; therefore, there is a larger incentive to take risks on these purchases and allow higher-return-probability instances to go through unwarned. The opposite is true for low-cost items, which are more readily classified a 1 if there is any reasonable doubt about the customer returning the item. To decide which classification to make for a given purchase, the probability value from the model prediction is compared to the cost-minimal threshold (Tau) for the particular item. If the prediction is greater than Tau, the customer is classified as 1 and shown the warning.

To this end, maximisation of AUC is still an effective evaluation method for models. This is because the more accurate a model's predictions are, the better they will perform when compared to cost-minimising Tau values. High- or low-risk purchases will be well classified at the appropriate value between 0 and 1, and decision errors will be minimised. To be sure, however, that models are not producing estimates on a scale that is different than that of the Tau values, a calibration curve is constructed. In the case that model predictions range, for example, between .4 and .6 (Tau values being between 0 and 1), the scale of these probability ranges is not equal, making it ineffective to compare them for final classification decisions. The resulting calibration curve for the XGBoost and random forest models, however, show that the predicion scale is near perfectly situated between 0 and 1, providing further assurance of the models' quality.

Special Modeling Challenge (Ensembling)

While standalone XGBoost and random forest models perform quite well, the combination of these models' prediction output improves performance scores even further. For this business case, ensembling is performed by a custom-built, automated function to average the prediction outputs of models constructed on x cross-validation folds. Using cross validation on subsets of the known data, variance in the prediction is reduced when aggregating fold outputs; that is, fold models with evident under- or overfitting effectively cancel each other out for a less biased final prediciton.

Candidate models for ensembling include 5 neural networks, 9 XGBoost models, and 6 random forests. Over x folds, all models are trained and then run through an iterative averaging process, which finds the initial best model and averages this model with all others, at each step calculating the new AUC against the fold's test data. The averaged combination which produces a new, higher AUC score is put through the same process once more. This repeats until no improvements to AUC have been made. For some folds this is two iterations; for others it is upwards of 10. Once all fold training has been completed and the candidate models ensembled appropriately, prediction vectors for the 50,000 rows of unknown data (x vectors for x folds computed and stored parallel to model ensembling) are then averaged a last time to produce the final ensembled prediction vector.

In the first Kaggle submission, with ensembled predictions utilising 5 fold CV, the unknown data AUC score increased from 0.74105 to 0.74300. With 10 folds, AUC increased again from 0.74300 to 0.74314. Though only minor increases, these marginal improvements can mean significant revenue generation / cost reduction in a production environment and highlight the value of cross validation and model-average ensembling to reduce prediction variance and increase predictive accuracy.

Conclusion

Using the final model-ensemble predictions from 10-fold cross validation and the cost matrix Tau values for binary selection, 18,700 of 50,000 customers are predicted to be valid return risks, with the final AUC value of .74314 indicating a roughly 74.3% rate of correct classification. Upon deploying this ensembled model to the e-commerce webshop, to the degree that these 18,700 risk cases and the remaining non-risk cases are categorised correctly, risky transactions can be flagged, warning messages displayed, and losses on product returns minimised.

References

1) Stefan Lessmann, Kristof Coussement, Koen W. De Bock, Johannes Haupt. (2018). Targeting customers for profit: An ensemble learning framework to support marketing decision making. *IRTG 1792 Discussion Paper Series 2018.* p. 5-13.

2) An inumberable count of Stack Overflow, Stack Exchange, Cross-Validated, etc. forum threads.

3) Course slides & exercises.