Business Analytics and Data Science

Term paper

Thomas Siskos, 580726

March 7, 2019

1 Introduction

During online purchases customers often send back items they order. These returns are costly and because of the high competition in online retail it is not possible or highly inadvisible to pass on the costs of return shipping to the customer. Therefore, accurate predictions of product returns could allow online retailers to impede problematic transactions, for example by restricting payment options or by displaying a warning message and thus cut down on cost due to shipping.

Providing a suitable predictive machine to this problem is the aim of this analysis. To this effect, data about the online purchasing behaviour as well as the information if the item has been returned or not has been provided for a german online-retailer. For this paper multiple machine learning approaches have been examined and documented.

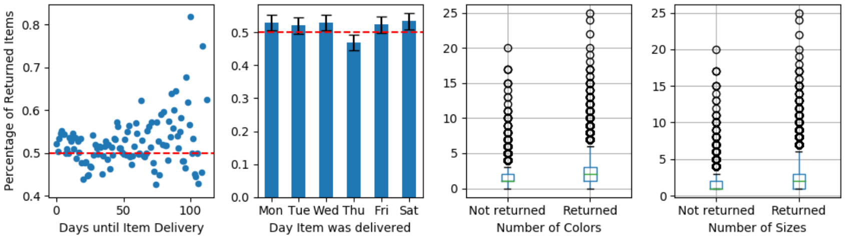
Section 2 contains a description of the data and the exploratory results results. Section 3 describes the actions taken in order to clean the data and a brief overview of the efforts taken during feature engineering. Section 4 specifies the different algorithms that were deployed and section 5 includes a succinct discussion of the results. Section 6 tries to minimize the costs of misclassifying an item directly by using a custom built and modified Genetic Algorithm. Finally, section 7 contains concluding remarks.

2 Exploratory Data Analysis

The data consists of 10000 samples and contains information on the item's color, the price, it's size and if it was returned by the customer or not. Additionally the data set encompasses information on the customers such as a unique identifier and the dates on which the item in question was ordered and delivered. With this information it is possible to retrieve the information on a complete order. In the following analysis an order is considered to be a collection of item orders that were placed by the same customer, on the same order date.

Figure 1 summarizes the initial findings through graphical evaluations of the data. It is apparent that a higher waiting time, i.e. the number of days that

Figure 1: Conditional Probabilities of orders containing a returned item



passes between ordering the item and it being delivered increases the likelihood of an item being returned. Additionally, it seems that items that are delivered on a Thursday on average get returned less. This difference is significant compared to all other days of the week, however the magnitude of this discrepancy is not substantial. Furthermore, the number of different sizes and the number of distinct colors in one order seem to differ slightly accross orders that contained a returned item compared to those who did not. Orders that were returned were both on average as well as in their 3rd quartile larger, and consisted of a broader array of colors. Nevertheless, in their lower end both distributions are virtually indistinguishable. This means that while there is a signal in the raw data it first needs to undergo a refining process before it can be used as a feature with a consequential predictive capacity.

3 Data Preparation

The data contained numerous missing values. Some of these were obscured by unorthodox codes, specifically for the delivery date it seemed that dates lying as far back as 1994 were being used to encode a missing value.

However, once encountered, these missing values were easy to impute by the mean number of days passed between the order and the delivery of the item for cases were the delivery date was available. When looking at the distribution of days between order and delivery it seemed more in order to use the median since it was heavily skewed, yet imputing by the mean seemed provide better results, changing the imputation method alone resulted in an increase in the Area Under Curve of two percentage points.

Similarly, for some users it was difficult to compute their age, since they either did not provide their day of birth or instead opted to provide implausible years of birth, such as being born in the beginning of the 20th century. Consequently, all years of birth lying farther back than 1926 were removed. Their age was imputed by first computing the mean difference of days between birth and registration, where they were available. This difference in days has been subtracted from the registration date of all users with incredulous birth dates.

The data contained also a large number of categorical variables which in

Table 1: Selection of engineered features

	feature	description
users	`tenure`	days between registration and order
items	`price-off`	discount compared to maximum item price
orders	`num items`	count of item IDs in order
	`days until delivery`	days between order and delivery
	`num sizes`	count of unique sizes
	`total value`	sum of all item prices in order
	`num colors`	count of unique colors
	`seq number`	enumerate order date per user
brands	`brand mean price`	average price of item's brand
state	`state mean delivery`	average number of days until delivery

turn contained numerous levels. Especially the items' colors involved some spelling mistakes and extravagant names for different shades of the same color. Both problems were solved by manually sifting through the various labels and summarizing the more detailed color names into broader categories. This way it was able to reduce the 85 initial colors to 14 unique levels in the cleaned data. For these densely populated categories it was possible to calculate the Weight of Evidence (Gough, 2007).

The items' sizes proved to be difficult to clean. Ideally one would want to extract categories like *small*, *medium*, *large* and while these are provided in some, they are not provided in the majority of cases. Instead there is a whole clutter of different sizes and without knowing the type of clothing only limited information can be extracted.

The starting point in cleaning the numerical labels in the item size was in identifying numerical subpatterns, for example the values between 1 and 14 are densely populated, then there is a break, where there are no occurences of the values between 15 and 17, this means that the interval $[1, 14]$ must represent one category which in this case these are likely to be hats. Following the same strategy, five other subgroups were identified. Next, each group was split into five uniform subintervals which represent the labels *XS, S, M, L* and *XL*. Additionally, through the use of regular expressions, it is possible to determine if items are pants, since they have exactly four numerical digits (two for the width and two for the length).

Figure 2: Feature Correlation Plot

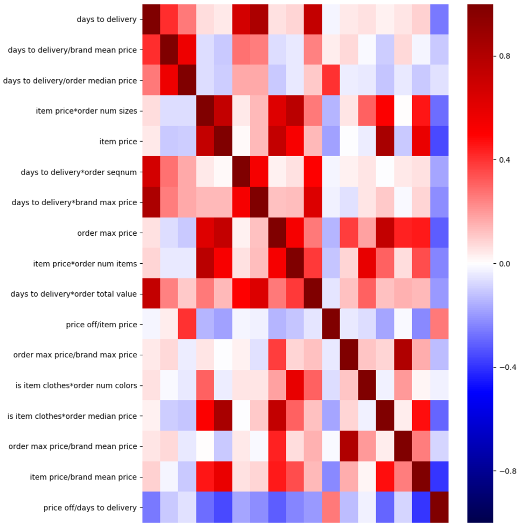


Table 1 contains a selection of the most important engineered features which are also used in similar works (Urbanke, Kranz, & Kolbe, 2015). Furthermore, all possible pairwise ratios and interaction terms were computed

Table 2: Parameter Spaces

	Parameters	Choices
Logistic Regression	Penalty	L1, L2
	λ, penalty parameter	uniform(0, 1)
Support Vector Machine	Kernel	linear
		radial basis function
		sigmoid
	Shrinking	True / False
	λ, penalty parameter	uniform(0, 1)
K-Nearest Neighbors	k	uniform(3, 15)
	weighting	equal / distance
	p	1 / 2 / 3
Random Forest	number of estimators	uniform(100, 15000)
	proportion of features used	uniform(0.2, 0.5)
	maximum depth	uniform(1, 100)
	minimum samples for split	uniform(8, 400)
	minimum samples per leaf	uniform(8, 400)
Feed Forward	architectures	up to 3 hidden layers
Neural Network	activation function	tanh
		ReLu
		sigmoid
		identity
	solver	adam
		Gradient Descent
Boosted Trees	number of estimators	uniform(10, 1000)
	maximum depth	uniform(1, 5)

where the most correlated features were discarded afterwards both for reasons of efficiency as well as numerical stability.

Finally, a Random Forest was trained, in order to extract the most important features out of the 163 that were generated. Subsequently the 16 most influential variables were picked based on their variable importance for further modelling, their correlation structure is summarized in figure 2.

4

Multiple different models and approaches were tried. Among which were Random Forests, Logistic Regressions, k-Nearest-Neighbor Classifiers, Support Vector Machines and simple feed forward Neural Networks.

A Random Forest is a combination of decision trees, where each tree is fitted with a random subsample from all cases as well as a randomly selected subsample of the features (Breiman, 2001).

As the name suggests, a K-Nearest-Neighbor Classifier computes the distance of a new sample compared to every sample of the training set and then selects the k closest cases, which are used to determine the new sample's label by a majority vote (Mucherino, Papajorgji, & Pardalos, 2009).

Support Vector Machines try to find the hyperplane defined by

$$f(x) = \beta^T x, \text{ where } |\beta^T x| = 1$$

that maximizes the margin between two classes in a high, possibly infinite dimensional feature space using the so called kernel trick. Maximizing the margin can be reformulated into minimizing a function $\mathcal{L}(\beta)$ subject to some constraints, such that

$$\min_{\beta} = \frac{1}{2}||\beta||^2 \quad s.t. \quad y_i(\beta^T x_i) \geq 1 \ \forall \ i$$

which is a Lagrangian optimization problem, whose solution provide the optimal parameters for the separating hyperplane (Hearst, 1998).

For determining the optimal parameters for each model the Python module `hyperopt` was used. The optimization routine deploys a Tree of Parzen Estimators in order to determine the best hyperparameters. A Tree of Parzen Estimator tries to maximize the so called information gain *EI*, which is defined as

$$EI_{y^\star}(x) = \int\limits_{-\infty}^{y^\star} (y^\star - y)p(y|x)dy,$$

where y is some real valued objective function $y^\star$ is some threshold of y and x is a set of hyperparameters. Mainly the Tree of Parzen Estimators splits the conditional distribution of the parameter vector in two parts

$$p(x|y) = \begin{cases} l(x) \text{ if } y < y^\star \\ g(x) \text{ if } y \geq y^\star. \end{cases}$$

Where it tries to avoid the distribution g leading to a value above $y^\star$ and favors the distribtuion l which tends to lead to values for y which are lower than $y^\star$ (Bergstra, Bardenet, Bengio, & Kégl, 2011). Table 2 shows the different parameters along with their prior distributions which were optimized.

However, most of these methods provided unsatisfactory results, and serve merely as a baseline. The most promising algorithms were a simple Feed Forward Network, the Random Forest that was used for determining the variable importance and the Gradient Boosted Trees.

5 Model Evaluation

Note that, while the Gradient Boosted Trees perform best on the test set, they also are most likely to overfit the training data and since this model has the highest discrepancy between train and test score it is not advisable to use it for the final predictions. One would much rather choose either the Neural Network or the Random Forest.

Again, between those two the Neural Network has the more favorable difference between the AUC of the train and test set and thus will most likely generalize better than the Random Forest.

Table 3: Selected Results

	Train AUC	Test AUC
Logistic Regression	0.620	0.617
K-Nearest-Neighbors	0.690	0.647
Support Vector Machines	0.598	0.596
Feed forward Neural Network	0.711	0.705
Random Forest	0.718	0.704
Gradient Boosted Trees	0.773	0.712

6 Minimizing Costs directly

Up until now only standard measures like the Area Under Curve (AUC) have been used in order to quantify and balance the particular cases of misclassification. However, in a business setting such as this differen faulty categorizations are typically associated with different types of costs. So naturally, it is desirable to operate on these directly in order to minimize them. Unfortunately, this problem is in its essence non-continuous and as such it is not possible to compute the gradients directly. This necessitates the use of non-standard optimization techniques (Goldberg, 1989).

One such technique is to use the predictive framework of the Logistic Regression in conjunction with a Genetic Algorithm. The Genetic Algorithm tries to emulate an evolutionary process, where in the first step candidate coefficient vectors $\beta \in \mathbb{R}^d, d \in \mathbb{N}$ are randomly sampled and ranked with respect to the cost which results from their predicted probabilities $p(y = 1|\beta)$ and a threshold c that serves as a decision criterion, where

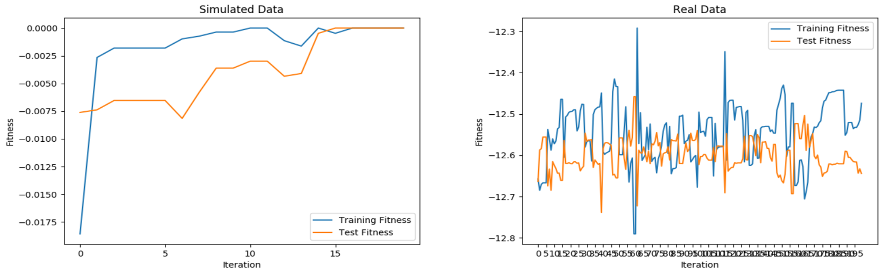
$$\hat{y} = \begin{cases} 1 & if\ p(y|\beta) > c \\ 0 & else \end{cases} .$$

The most successfull candidates are then allowed to propagate and be combined to form the new set of candidate values for β in the next iteration (Holland, 1992). An outline is given in pseudocode in algorithm 1. One problem with this approach lies in its tendency to overfit. In order to mitigate this, two minor modifications where added. First, a random sampling technique was used, where the fitness of each candidate in the population was computed only by a small

Algorithm 1 Genetic Algorithm

```
history ← []
for i in 1, ..., maxiter do
    # Initialize/Reset lists
    fitness ← repeat($-\infty$ , population_size)
    cutoffs ← repeat( $-\infty$, population_size)
    if random() < reset_probability then
        X_train, y_train, price_train ← resample()
    end if
    for j, $\beta$ in enumerate(population) do
        y_pred ← predict_proba(X_train,$\beta$)
        fit, cut ← get_fitness_and_c(y_pred, y_train, price_train)
        fitness[j] ← fit
        cutoffs[j] ← cut
    end for
    # Select the most fit individuals
    fit_idx ← argsort(-fitness)
    parents ← fit_idx[0:num_parents]
    # Overwrite population
    population ← population[parents]
    # Crossover
    while length(population) < num_parents do
        parent_a, parent_b ← sample(population, 2)
        candidate ← crossover(parent_a, parent_b)
        if random() < mutation_probability then
            candidate ← mutate(candidate)
        end if
    end while
    # Store results
    best ← argmax(fitness)
    history[i] ← population[best]
end for
return argmax(history)
```

Figure 3: Results of the Genetic Algorithm



(a) For the simulated test case it is easy to find an optimum that generalizes well.

(b) For the real data considerable overfitting is to be expected.

fraction of the available training data. This sample will be randomly redrawn in each iteration, which means that only solutions that generalize beyond many training sets have a chance to be propagated in the long term. Additionally this speeds up the run time of the algorithm and, fortunately, leads also to a better generalization overall (Gonalves, Silva, B. Melo, & Carreiras, 2012). Secondly, for the final result of the routine, instead of using the candidate β that had the best fitness on the training data, the candidate is used that proved to have the highest fitness, i.e. the lowest cost on its respective out-of-bag-samples.

These modifications mean that it was not possible to use an out-of-the-box implementation for the Genetic Algorithm and it was necessary to write a custom variant. On the positive side this allows for a very tailored approach to the problem. For example, when determining the fitness of a parameter vector β the Logit Model will only provide probability estimates and the parameter c for determining the predicted label of a particular case still needs to be tuned. In the custom implementation the fitness is evaluated for a particular realisation of β on a whole grid of values for c. Afterwards the threshold for this particular β is chosen such that it maximizes the fitness - as opposed to fix a threshold beforehand as a sort of hyperparameter.

In order to test the customized algorithm a simulation study was used. A data matrix X was randomly initialized and a vector of known, 'true' coefficients β_{true} was used to calculate deterministic true labels y_{true} with the threshold $c_{true} = 0.7$ much in the same way as described above. The results of the simulation are described in the left part of figure 3. The algorithm was able to find an approximate solution that was a scaled version of β_{true} with a different threshold which provided a cost on the out-of-bag sample that was sufficiently close to zero. Having confirmed the reliabilty of the handcrafted routine in such a way it is possible to further assess the cost on the real data.

Furthermore, in order to reduce the dimensionality of the solution and thus to facilitate the search of the algorithm, the original feature space was decomposed into it's 4 Principal Components. Due to the long run times of the algorithm, it was too costly to deploy a traditional grid search or a search by a tree of Parzen Estimators, with sufficient trials. Instead a grid of numerous configurations has been determined and only a random selection of this grid has been evaluated. The run that provided the best average cost per item on its respective out-of-bag samples was used for the final predictions.

7 Conclusion

References

Bergstra, J. S., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 24* (pp. 2546–2554). Curran Associates, Inc. Retrieved from `http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf`

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. Retrieved from `https://doi.org/10.1023/A:1010933404324` doi: 10.1023/A:1010933404324

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning* (1st ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Gonalves, I., Silva, S., B. Melo, J., & Carreiras, J. (2012, 04). Random sampling technique for overfitting control in genetic programming. In (p. 218-229). doi: 10.1007/978-3-642-29139-5_19

Gough, D. (2007). Weight of evidence: a framework for the appraisal of the quality and relevance of evidence. *Research Papers in Education*, *22*(2), 213-228. Retrieved from `https://doi.org/10.1080/02671520701296189` doi: 10.1080/02671520701296189

Hearst, M. A. (1998, July). Support vector machines. *IEEE Intelligent Systems*, *13*(4), 18–28. Retrieved from `http://dx.doi.org/10.1109/5254.708428` doi: 10.1109/5254.708428

Holland, J. H. (1992). Genetic algorithms. *Scientific American*, *267*(1), 66–73. Retrieved from `http://www.jstor.org/stable/24939139`

Mucherino, A., Papajorgji, P., & Pardalos, P. (2009, 01). In *Data mining in agriculture* (Vol. 34, p. 83). doi: 10.1007/978-0-387-88615-2

Urbanke, P., Kranz, J., & Kolbe, L. (2015). Predicting product returns in e-commerce: The contribution of mahalanobis feature extraction. In *Icis.*