Business Analytics and Data Science

Term paper

Thomas Siskos, 580726

March 1, 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.

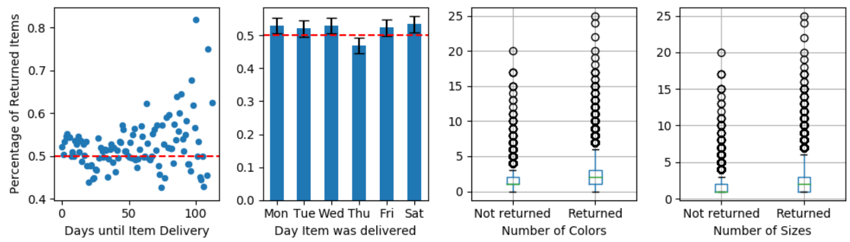
Section 2 contains a description of the data and the exploratory results results. Section 3 describes the actions taken in order to clean the data and a brief overview of the efforts taken during feature engineering. Section 4 specifies the different algorithms that were deployed and section 5 includes 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 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 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 significantly 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

Figure 1: Conditional Probabilities of orders containing a returned item



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 their 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. 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 ones. Consequently, all years of birth lying farther back than 1926 were removed

Figure 2: Feature Correlation Plot

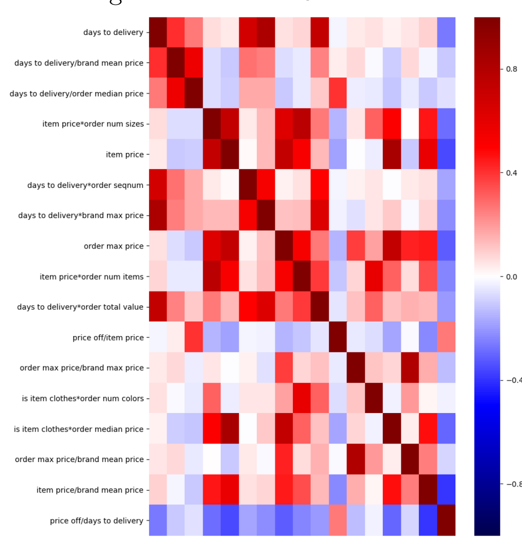


Table 1: Selection of engineered features

	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

and the age of these users was imputed by the difference in days between registration date and the birth date, of the valid users which was subtracted from the registration date of the incredulous ones.

The data contained a large number of categorical variables which in 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. Through the use of regular expressions, for example, 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).

Table 1 contains a selection of the most important engineered features. Furthermore, all possible pairwise ratios and interaction terms were computed where the most correlated features were discarded afterwards.

Finally, a Random Forest was trained, in order to extract the most important features out of the 163 that were generated. Subsequently the 20 most influential variables were picked based on their variable importance for further modelling.

4 Model Tuning and Selection

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

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

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, mostof 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.

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)

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

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.

6 Minimizing Costs directly

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

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

Mucherino, A., Papajorgji, P., & Pardalos, P. (2009, 01).

Algorithm 1 Genetic Algorithm

```
history ← []
for i in 1, ..., maxiter do
    # Initialize/Reset lists
    fitness ← repeat(−∞ , population_size)
    cutoffs ← repeat( −∞, population_size)
    if random() < reset_probability then
        X_train, y_train, price_train ← resample()
    end if
    for j, β in enumerate(population) do
        y_pred ← predict_proba(X_train,β)
        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)
```

In *Data mining in agriculture* (Vol. 34, chap. k-Nearest Neighbor Classification). doi: 10.1007/978-0-387-88615-2