Machine Learning in Marketing

Coupon Optimization using a Convolutional Neural Network

Iliyana Pekova, Georg Velev, Gleb Zhidkov, Mirsad Sukilovic

Humboldt-University

March 21st, 2021

We develop an approach to optimize the choice of personalized discount coupons for a large retail product assortment. Our two-step solution consists of a multiple input convolutional neural network (CNN) to predict purchase probabilities and an algorithm to assign revenue-optimizing individualized coupons. Our CNN model outperforms two naive benchmarks, whereas our personalized coupon optimization algorithm leads to a revenue uplift of 23.5% compared to a setting where no coupons are provided, and of 28.5% compared to coupons assigned at random.

Github project repository
Data

Business problem

The advances in technology and the ease to collect vast amounts of behavioral customer data have accelerated the shift from mass to individualized marketing. The large products assortments and customer bases in retail require scalable models that can easily learn complicated structures of often unlabeled data.

Discount coupons remain one of the major methods to stimulate customer retention and improve revenues. Yet the redeem ratio rarely exceeds 1%, and it is likely that coupons are redeemed for products that would have been purchased anyway, ultimately leading to unnecessary marketing expenses. Coupon personalization is a possible solution to this problem, as knowledge about the past purchase behavior of a customer can be utilized to optimize the coupon assignment, generating additional revenue that is lost when coupons are assigned at random or using heuristics.

In this report, we present (1) a convolutional neural network that predicts purchase probabilities based on personal past purchase and coupon redeeming history data that is then used to (2) generate personalized discount coupons optimized to maximize the expected (individual) revenue. The expected revenue per product per shopper is defined as the expected product purchase probability times product price (accounting for offered discount, if any).

Purchase probability for a given product by a customer is assumed to depend on her past purchases of this and other products as well as on coupons offered to this customer for the current and previous periods of time.

State of art & coupon effects

Despite the recent trend for individualized marketing, the vast majority of coupons (and other types of discounts) is distributed based on the supply chain rather than the customer perspective. In other words, products are promoted based on managerial heuristics, e.g. coupons are distributed for those products that are in excess stock or are only facing a low demand. While such promotions may reach their desired effect (of reducing the stock of the discounted product), they are unlikely to improve customer retention or revenue, as they usually lack any aspects of personalization, and can be considered random from the customer perspective.

Customer coupons that are *actually* assigned at random are likely to be a less likely phenomena in real business applications, as any business heuristic (even the one that is ultimately harmful for business success) would be preferred by decision makers to a random discount assignment. If such random coupon generating mechanisms exist in retail, they—by definition—are not considering any customers' preferences and can potentially, similar to when managerial heuristics are applied, lead to a decreased rather than increased revenue.

A negative revenue uplift can occur for multiple reasons. The most prominent example are discount coupons provided for products that a customer is intending to purchase anyway. Such coupons may be beneficial for customer retention and satisfaction ("I was planning to buy my favorite yoghurt and I got a discount, now I will definitely visit the XYZ store!"), they may have countering effects on the profit of the retailer, as the marketing expenses (coupon discount) do not induce additional revenue.

Cross-product and cross-coupon effects may play an important role as well. If a coupon is granted for a product X (e.g., cookies), the customer may refrain from purchasing some substitute product (e.g., the more expensive chips that the customer buys on a recurring basis—this is often referred to as *within-category effect*). For some other products, the cross-product effect may have an opposing direction (consider getting a coupon for pesto and additionally buying some spaghetti—is referred to as *cross-category effect*).

Usually, a product that has been discounted faces a higher demand (*own-coupon effect*). In rare cases, the effect may be countering, and lowered prices may push customers away, e.g. because some deficiency will be assumed such that customers will refrain from a purchase altogether.

Most of the coupons are never redeemed, which may often be simply the case because those coupons were not for something the customer would consider to buy. To offer coupons that are indeed relevant to the shopper, the coupon-generating algorithm should have a sufficient understanding of the customer's needs and preferences, which usually evolve over time (*stock effect*), with certain products only required periodically.

For example, a recently purchased washing powder will drastically reduce the effect an assigned coupon for even more washing powder may have. At the same time, a coupon for fabric softener as a related product may now significantly increase the likelihood that this product will be purchased ("This is exactly what I need but forgot to buy yesterday!").

Finally, coupons given to but not redeemed by the customer in the past may raise the awareness of the customer of a certain product or product category, ultimately leading to an additional, non-intended purchase ("Alright, I will try this one out after all...").

Input Data

Our input data for modeling consists of three matrices, containing historical data about products and coupons. The first input is a JxT matrix, capturing the purchase history of a customer i for every week in a given period T for every product j of J=250 products. The cell values are binary with 1 if customer i purchased product j in week t and 0 if they did not. The purchase history only includes data from T=5 most recent weeks to the week we wish to generate a prediction for, since they are likely to have the strongest impact on the purchase behavior for the next week.

The second input is a Jx5 matrix, containing purchase information of the 25 weeks prior to the 5 most recent weeks. This implies that the first label the model will see during training will be the one of the 31st week. The 25 weeks are split in 5 periods and the average purchase frequency per product is computed for each one of these, meaning that the cell values are in the range [0, 1].

The third input is a Jx6 matrix, containing the size of the coupon discount for each product of vector J for each one of the most recent 5 weeks and for the 6th week, meaning that we have considered both historical coupon data and data for the week to be predicted. Information for past coupons is needed to enable the model to learn the relationship between shopper behavior and availability/lack of discounts, or how discounts impact the purchase behavior. The data for the week that will be predicted is of importance, since this respects the assumption that each coupon can only be used in the week it has been received.

Apart from the three input matrices the model is also fed a vector holding binary labels, indicating whether a product j was purchased by customer i in week t. **Figure 1** visualizes which data matrices will be fed into the model first. The corresponding labels to this input data are the purchases for each product (not) done in week 31. The model will learn what shopper behavior per product in week 31 was, based on the three input sources.

Figure 1: First sample of input data for customer i

1st input matrix: purchase history for customer i

	T = 26-30 (incl.)				
	26	27	28	29	30
J1					
J2					
J3					
...					
J250					

2nd input matrix: periodical purchase frequencies for customer i

	t = time between week 0 and week 25 (incl.)				
	0--5	6--10	11--15	16--20	21--25
	avg freq	avg freq	avg freq	avg freq	avg freq
J1					
J2					
J3					
...					
J250					

3rd input matrix: periodical discounts for customer i

	t = time between week 26 and week 31 (incl.)					
	26	27	28	29	30	31
	discount	discount	discount	discount	discount	discount
J1						
J2						
J3						
...						
J250						

Model architecture

In our modelling approach, we closely follow the model introduced in "*Product Choice with Large Assortments: A Scalable Deep-Learning Model*" (2020) by Sebastian Gabel and Artem Timoshenko.

Their model incorporates three input layers which encode customer's recent purchase history, purchase frequencies (for all known time periods), and coupons assigned to the customer for the upcoming time period. These inputs are pushed through an

encoder-decoder layer such that the model is capable of learning the underlying stock, cross-coupon, and own-coupon effects.

Different to their model, our implementation consumes more input information, including the recent purchase history (for 5 most recent weeks), the extended purchase history (with purchase frequencies of 25 preceding weeks, aggregated into 5 columns), and the recent coupons offered to the shopper (for 5 most recent weeks and for the current week for which the prediction of purchase probabilities is made). These adjustments allow the model to learn more about the temporal purchase behavior of the customer both in terms of her purchases and in terms of coupons recently offered.

We implement a convolutional neural network (CNN) with one dimensional layers in order to generate predictions of customer-specific purchase probabilities. The reason for choosing a CNN model is that CNNs apply a sliding kernel over the sequences in each mini-batch which enables the model to learn interrelations across multiple features in each sequence. When applying a one dimensional CNN model, the kernel slides along the product dimension (y-axis).

This implies that the CNN model will learn cross-product patterns in the data as the kernel will slide along the product dimension. **Figure 2** visualizes the sliding of the kernel in each sequence when the input data is the purchase history per shopper.

The CNN layers generate feature maps as a result of the sliding of the filters. The feature maps are applied to all weeks and learn the relationships across different products.

Figure 3 visualizes our proposed model architecture. It can be seen that the model has 3 input layers for each data source. The input layers are followed by CNN layers with 18 filters each. We set the number of filters to a rather small value due to the high computational cost of the training process.

Figure 2: Sliding Kernels

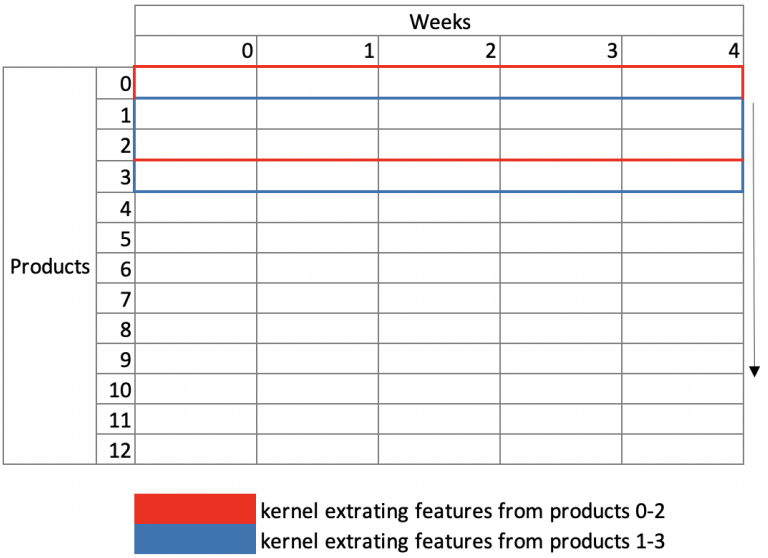
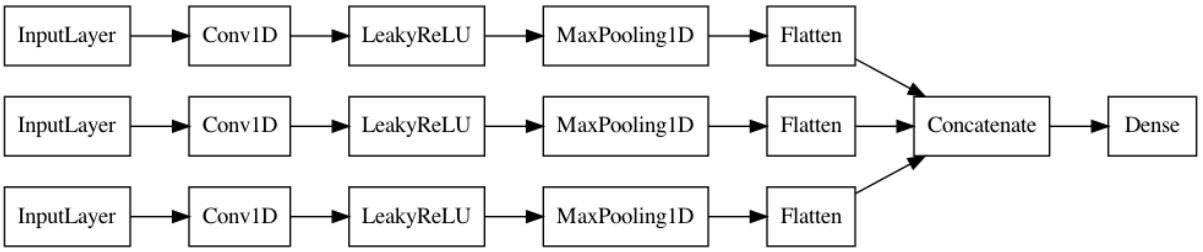


Figure 3: CNN architecture



The CNN layers pass the generated feature maps to LeakyReLU layers. We select LeakyReLUs as activation layers to avoid the so-called "dying ReLU problem" which leads to a vanishing gradient descent in the negative region of the learnable parameters (weights). By contrast, LeakyReLU multiplies negative values by a small value called alpha, for example 0.001, which prevents the gradients of the weights in the negative region to be equal to 0. This way, both the negative weights as well as the positive ones are continuously updated during the training process.

The LeakyReLU layers are followed by Max-pooling layers. Max-pooling layers extract the features from the feature maps where the neurons have fired the highest values and reduce the dimensionality of the feature maps. This way, Max-pooling layers aim to prevent the model from overfitting the data by concentrating only on the most important locations in the feature maps, meaning the locations with highest values.
The Max-pooling layers are followed by Flatten layers which stretch the learned features into vectors as a preparation for the final Dense layer. The flattened features are then concatenated and passed to the Dense layer which applies the sigmoid activation function in order to squash the values between 0 and 1.

The Dense layer has 250 neurons which is determined by the number of products in the dataset at hand. This implies that the Dense layer generates shopper-specific purchase probability predictions for each product in a given time period.

The compiled CNN model has a total of 1,675,168 trainable parameters.

Model evaluation

We use weeks 30–79 to train the model (weeks up until 30 are lost as 30 weeks of history are required and no padding has been applied), and we validate its performance by predicting purchase probabilities for weeks 80–89. To reduce the execution time, we only use a subset of 6000 shoppers and 10 epochs for training in mini-batches of 10 shopper-week combinations, yet this is sufficient for the model to grasp the data structure as discussed in the previous sections.

There are multiple meaningful reference solutions for the problem at hand. For example, Gabel and Timoshenko (2020) benchmark their model using a binary logit model, LightGBM (a gradient boosting decision tree implementation), and a

hierarchical multinomial logit model. Nonetheless, all of these models require additional data preparation steps.

Due to the time constraint, we only validate our model against two naive benchmarks: randomly generated purchase probabilities (between 0 and 1) and a fixed purchase probability of 0.04 which approx. equals to the purchase probability in the simulated dataset used to train the model. Our model outperforms both benchmarks, with the binary cross-entropy loss for train and test data reported in **Table 1**.

Table 1: Achieved loss of our model and benchmarks

	Our model	Random prediction	Fixed value
Train set	0.07315	0.99986	0.13828
Test set	0.08931	1.00022	0.13815

Coupon optimization

Optimizing coupons with respect to the expected revenue is a computationally expensive task. When having a rather small product set of 250 products, there exist nearly 8 billion combinations of how 5 coupons can be assigned to a single customer. As the purchase probabilities depend on the assigned coupons, we would have to use the previously trained model to predict them for every possible coupon combination.

As this is clearly not feasible, we apply a greedy search with an early stopping heuristic over possible coupons and their discount values. The proposed algorithm to assign optimal, individuals coupons for a given week and shopper can be summarized as follows:

- Coupons with the highest discount (here: 30%) are assigned to each product, creating a mini-batch of size 250. Expected revenue based on model prediction probabilities is calculated for all options in the mini-batch, and the revenue maximizing coupon is selected (if such exists). This is repeated for the next discount value. If the lower discount value does not allow for a higher revenue, iteration is stopped early.
- The previous step is repeated for each coupon that is to be assigned to a shopper (that is, 4 times), keeping the previously selected coupon(s) without changes so that cross-coupon effects are accounted for.
- If no revenue improving coupon can be assigned at a given step, a random coupon is generated.

Coupons that are assigned earlier are therefore associated with a higher profit uplift (because cross-product relationships are symmetric, and no assigned coupon can uncover a more revenue-improving coupon for a later algorithm iteration, as this coupon would be selectable itself at an earlier stage). Because higher discount values usually are associated with higher customer response, the algorithm will stop early in most cases, further reducing the computational requirements.

Five coupons were optimized for week 90 for each of the first 2000 shoppers. To validate that results of the coupon optimization are feasible, the expected revenue when coupons are optimized is compared to the one when coupons are random, and when no coupons are assigned. The results are reported in **Table 2**.

Table 2: Achieved revenue uplift of our model and benchmarks

	No coupons	Random coupons	Optimized coupons
Average expected revenue per customer	4 487 €	4 251 €	5 541 €
Revenue uplift %	—	– 5.25 %	+ 23.50 %
Discount distribution 30% 25% 20% 15%	—	 2514 2508 2463 2515	 9312 513 154 21

Clearly, individually optimized coupons allow for a significant expected revenue profit uplift (which already accounts for the coupon discounts) compared to randomly assigned and to no assigned coupons. The coupon value distribution is skewed towards higher discount values, which suggests the belief of the model that customers are reacting much more elastically to higher discounts.

One further insight is that revenue-maximizing coupons were often assigned to nearby products, e.g. shopper 3 has been assigned coupons for products 119, 118, 35, 36, and 34. As we know from the basket data analysis (homework assignment 4), products are ordered by their (simulated) category, so that shopper 3 received coupons for five products likely belonging to only two product categories. This result can be explained as follows: when a customer receives several discounts for the same category, they become more persuaded to redeem one or more of them, hence increasing the expected revenue.

Discussion and limitations

In this study, we applied a multiple input convolutional neural network (CNN) to predict purchase probabilities based on past purchase and coupon history. Our model is capable of learning the data generating process and the multiple cross-products, cross-coupons, and cross-time effects that impact purchase decisions in a retail setting.

A CNN is a rather non-conventional solution for this problem, especially given that—different to image data—there is no straight-forward relation between data points along the product axis for purchase history data. In other words, products can be sorted in any way possible, making it much more computationally expensive for the model to learn the underlying relations within and between the different product categories. This problem should be at least partially mitigated by the dense layer at the head of the model.

Our model implementation does not require a pre-aggregation of the data, with input matrices constructed on-the-fly by the basket data streamer. Our implementation refrains from using pandas to pre-process data, as it has proven to be a major bottleneck when designing the pipeline. Overall, both the CNN model and the coupon optimization routine are sufficiently scalable to large amounts of data; both pipeline steps can be run on a distributed infrastructure with only little adjustments to the codebase required (out of the scope of this project).

There are several further limitations to our approach. First, no hyperparameter tuning has been performed, and the CNN model was trained on a limited number of data. Better fit results could possibly have been achieved with a different model configuration and with more data seen by the model. Second, our data streamer is designed such that it stores all shoppers in the same file, reading them row by row. A more scalable approach would rely on separate files for each shopper. Third, product prices are assumed to not impact the purchase probability, which is a rather unrealistic assumption. Finally, our model should be evaluated against other industry benchmarks—due to the time constraint, we only compare its performance with two naive benchmarks, which our model outperforms.

Our coupon optimization algorithm achieves a strikingly good results of nearly 30% revenue uplift, if compared to the revenue obtained when coupons are assigned. This result either implies that our model has done an incredibly good job at learning the preferences and needs of the (simulated) customer base, or that it is biased towards certain products or coupon effects. The actual revenue uplift for future time periods will likely be smaller, yet it is expected to be well above zero.

It is not possible to evaluate the accuracy of the coupon optimization algorithm on the past data, as no counterfactual can be modeled. In other words, it is not possible to measure the effect of alternative coupons assigned in the past, as the actual purchases (which have already been observed) depend on the actual discounts received by customers in the past.

Nonetheless, the evaluations results suggest that our CNN implementation delivers meaningful prediction results that are based on the true data generating process, accounting for stock, coupon and cross-product effects (with most predicted purchase probabilities approaching zero, and the expected revenue per customer within a reasonable bandwidth as compared to the revenues in the past weeks). Our coupon optimization algorithm efficiently generates personalized coupons that significantly improve the expected revenue uplift given the expected product purchase probabilities generated by the CNN model.