Instacart Customer's Next Basket Prediction

Robert Ciesielski (mentored by Dipanjan Sarkar)

Executive Summary:
We model customer's next backet prediction on a all-user and a per-user bases using the Instacart dataset and various recurrent neural network units convoluted with a dense layer perceptron for product classification, followed by probability ranking for product selection. Our study shows that individual per-user models perform better than one model for all users. While recurrent neural-networks have overall good performance, the model's predictive power may be limitted by the product ranking step. Results are compared to a baseline model defined as a copy of the customer's previous order.

Contents

1. Introduction

Instacart[1] is a company that offers their customers online grocery shopping from participating retailers. In 2017, Instacart released to the Kaggle community[2] an anonymized dataset corresponding to a sample of over 3 million grocery orders of more than 200,000 customers, asking the community to predict which previously purchased products will be present in a customer's next order. An efficient prediction of short-term customer behavior has a twofold advantage. On the one hand, it enhances the user experience on the Instacart's website (a recommender system idea). In addition, it allows the company to improve its supply chain, reduce the product delivery time, minimize the waste of fresh short-lived products, etc.

[1]https://www.instacart.com
[2]https://www.kaggle.com/c/instacart-market-basket-analysis/overview

The most successful approaches in the Instacart Kaggle challenge [1] exploited the idea of reducing the problem to pairwise (customer-product) binary classification, followed by product ranking using classifier output probabilities. The classification was predominantly performed using gradient boosting models (such as XGBoost, LightGBM, or CatBoost) and a limited number of customers' previous orders. Deep learning approaches were not very popular in this competition, only two documented projects exploited models based on recurrent neural network (RNN). In the solution described in Ref. [2] (the 3rd place on the leaderboard) RNN and all the customer's previous orders were used for feature extraction at the next-order level, subsequently fed into a LightGBM classifier to obtain customer-product probabilities. The other approach [3] (the 43rd place on the leaderboard) used a combination of only two adjacent orders at each time step of RNN (the current and the previous order) and product embeddings for each user and each time step.

In the following project, we explore models based on RNNs using all the customer's previous orders and one-hot encoding for the products purchased by a given customer. We investigate basic RNN architectures, as implemented in the Keras library, including single- and bidirectional Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) models, convoluted with a dense layer as a classifier, and compare their performance to a simple baseline model constructed as a copy of the customer's previous orders.

2. Dataset

The data is available on the Kaggle platform as seven CV files. Fig. 1 shows a schematic representation of the tables in these files. The master table, shown on the right-hand side of the figure, contains information on the orders purchased by about 206 thousand users. For each user, there are at least three and at most 98 prior orders. The last order is marked either as a train or a test order, to be used for model training or model prediction, respectively. There are about 131,000 and 75,000 users in

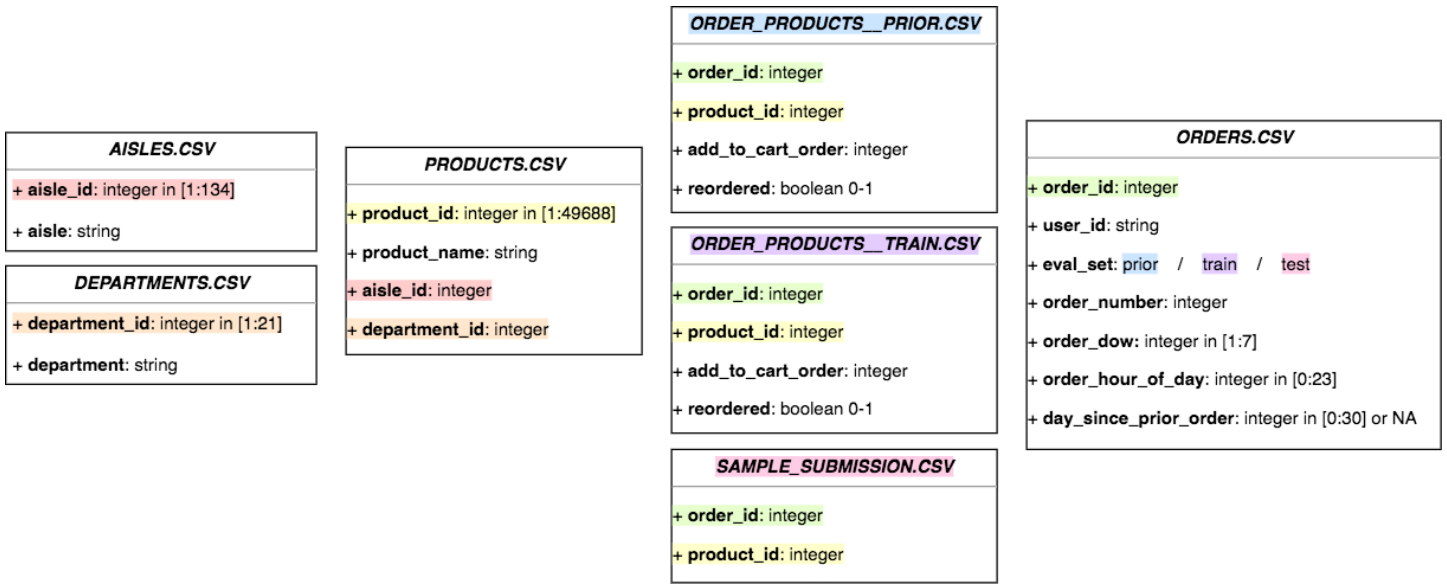

Figure 1. The layout of the data structured in seven tables. Different colors depict variables present in more than one table, which serve as links between the tables. Taken from Ref. [4].

the train and the test set. The table also contains time-related information about orders (see the next section).

The list of products in each order, the sequence they were added to the cart, and the information if a given product was ordered in the past (reordered product) are given in separate tables, separately for prior and train orders. The table with test orders has no product information and is provided as a template for model predictions. The remaining three files contain names of products and corresponding isles and store departments.

3. Exploratory Data Analysis

The number of orders purchased by each user is shown in Fig. 2 (top). The distribution peaks at the minimum number of 4 orders, then falls exponentially until the cut-off of 99 orders. The mean of the distribution is 17 and the median is 10 orders. Fig. 2 (middle) shows the distributions of the number of products in each order (blue histogram) and the number of products that were already bought by a user in the past (reordered products, yellowish histogram). The plot is cut off at 50 products, but the highest number of products extends to 145 and 130, respectively. The mean of the distributions is 10.6 and 6.8 products, respectively, and the average fraction of reordered to all products in an order amounts to 0.6. How much does the number of purchased products vary between orders for a particular user? This question is addressed in Fig. 2 (bottom), which shows a 2-dim histogram of the standard deviation vs the mean number of products averaged over all the orders purchased by a user, for the reordered products. The red line is a result of the fit with a linear function and is shown to guide an eye. The variation from order to order is rather large, the standard deviation is roughly 0.4 of the mean.

Fig. 3 shows time-related information about orders. Namely, the hour of the day and the day of the week when orders were

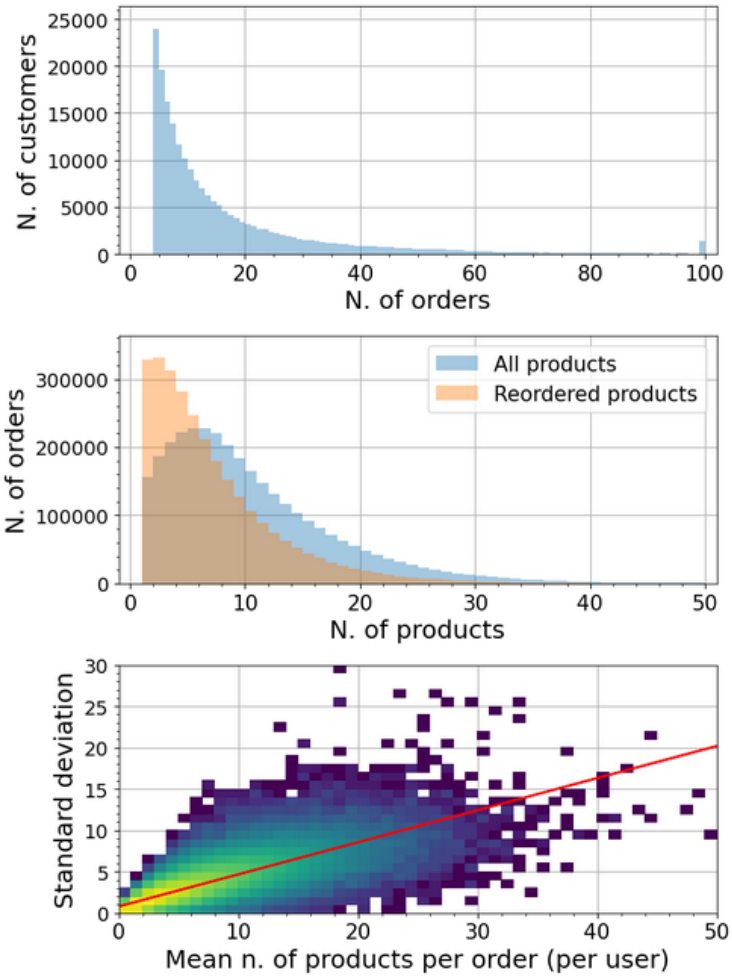

Figure 2. Number of orders per user (top), number of products and reordered products per order (middle), standard deviation vs mean number of reordered products in all orders (per user).

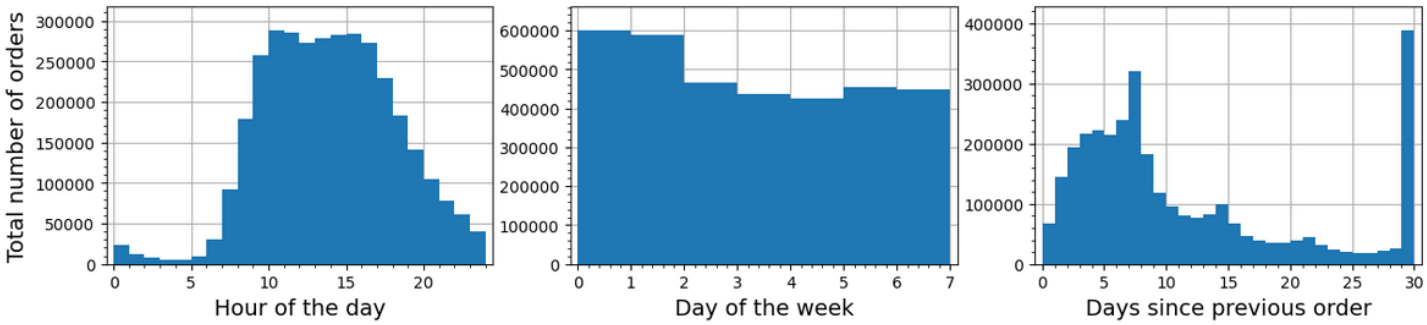

Figure 3. Time-related order information: the hour of the day (left) and the day of the week (middle) when a user placed an order, and the number of days since a previous order (right).

purchased, and a time difference in days between a current and a previous order. People are most likely to order during working hours (9am to 4pm) and orders are placed uniformly over the week, except for the first two days (Sunday and Monday?) when there are about 25% more purchases. In the last plot, the underlaying continuous spectrum has a maximum at about 4 days and a long tail (cut off at 30 days), on top of which there are peaks at 7, 14, and 21 days since previous order, which may indicate automatic recurring orders. We checked that the average number of purchased products depends only very weakly on these time-related variables (not shown).

Overall, there are about 50,000 products available in the store. Which of them are the most popular? Fig. 4 presents a frequency plot of the top-20 most ordered products, and shows that the most popular products are fruits and vegetables belonging to a healthy diet, e.g., potassium-rich bananas, avocados, and spinach, as well as berries, apples, lemons etc. The most popular are bananas,

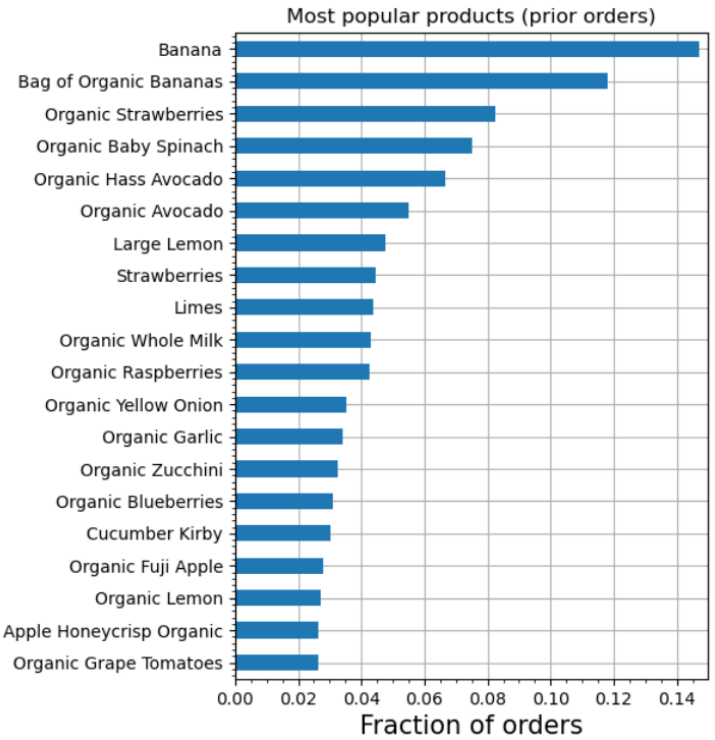

Figure 4. List of the top 20 most frequently purchased products.

purchased in more than 14% of orders. Together with organic bananas (2nd most popular), they are present in about 25% of orders. 14 out of top 20 products are organic.

4. Metrics, mean f1 score

A suggested metric for the model evaluation is the mean f1 score, defined as an arythmetic average of the individual user f1 scores. Each user's f1 score is calculated as a harmonic mean of precision, P, and recall, R, using the standard formula:

$$\mathrm{f_1} = \frac{2}{\mathrm{P}^{-1} + \mathrm{R}^{-1}}. \tag{1}$$

Given two lists of products, a true and a predicted one, P and R are defined as:

$$\mathrm{P} = \frac{\mathrm{N}^{\mathrm{PT}}}{\mathrm{N}^{\mathrm{P}}}, \quad \mathrm{R} = \frac{\mathrm{N}^{\mathrm{PT}}}{\mathrm{N}^{\mathrm{T}}} \tag{2}$$

where N^{P} and N^{T} are the number of products in the predicted and true lists, respectively, and N^{PT} is the number of common products in both lists. For two empty lists $P = R = f_1 = 1$, if one of the two lists is empty $P = R = f_1 = 0$, while non-empty lists receive $P, R, f_1 \in [0, 1]$

5. Baseline model

Before diving into machine learning modeling of customer next basket prediction, we construct a simple baseline model by using information available in his/her previous orders.

5.1 Copy of the customer's n-th last order

One could expect that returning customers have a set of products that they purchase regularly. In such a case, a hard copy of the user's last or the second last order would be a reasonable estimate of the next basket content. Fig.5 (left) shows the mean f1, mean precision, and mean recall scores evaluated by comparing the content of the user's future order (train sample) with a copy of his/her n-th last order. The value of n is shown on the horizontal axis and varies from 1 to 19. The highest mean f1 score of 0.326 is obtained if only the last order is taken into account.

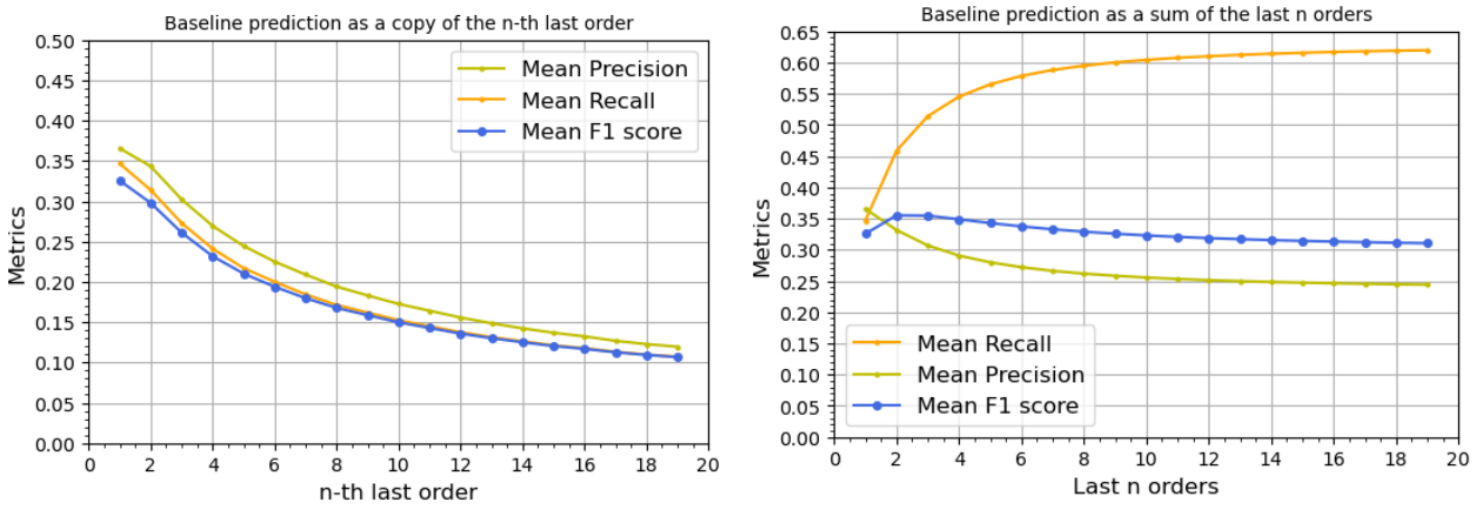

Figure 5. Mean f1 (blue), precision (yellow), and recall (orange) scores for the baseline model defined as a copy of the n-th last customer's order (left) and a sum of reordered products from the last n customer's orders (right).

5.2 Sum of the customer's last n orders

Can we do better? Will adding up the last n orders increase the model's predictive power? In Fig.5 (right) the scores are evaluated again, this time by comparing the future order with a sum of the last n orders. An interesting observation is that at a higher number of orders, recall reaches a plateau of about 0.6, while precision of 0.25. This means that when all orders are taken into account, approximately 2/3 of the products from the actual future order are predicted correctly, but only 1/4 of the predicted products are in the actual order[3]. The highest f1 score of 0.355 (recall $\simeq$ 0.45, precision $\simeq$ 1/3) is obtained for the sum of the last 2 orders. We will use it as a reference to be compared to ML models in the next sections.

5.3 Metrics stability

Fig.6 presents the mean f1 score calculated by comparing the future order with the sum of the last two orders for groups of 10,000 consecutive users in the data, showing that the score is stable over the entire dataset. As our intention is not to participate in the Kaggle competition, but rather to evaluate the performance of the RNN models, this feature will allow us to use only a part of the dataset to obtain a representative result.

[3]Morenover, the presence of recall plateau puts an upper limit of the mean f1 score that may be obtained for this dataset: P $\simeq$ 0.6, R=1 $\rightarrow$ $f_1 \simeq 0.75$.

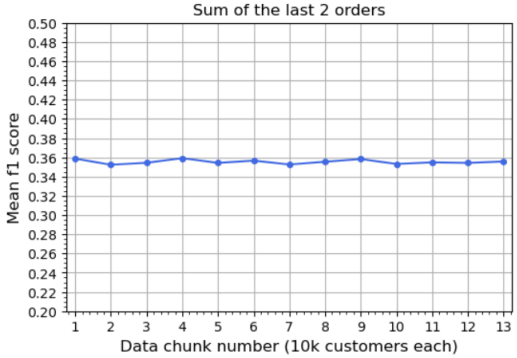

Figure 6. Mean f1 score for batches of 10,000 users in the data.

6. Recurrent neural network (RNN) models

It's time to recall basic recurrent neural network architectures for sequential data.

6.1 GRU

The input to the GRU (Gated Recurrent Unit) cell is the data at the time t (x_t) and the cell's hidden state from the previous time step (h_{t-1}) The flow of the information within the cell is controlled by the update gate (u), the reset gate (r), and a candidate hidden state ($\widetilde{h}$), as given by the formulae:

$$\begin{aligned} u &= \sigma(W_u \cdot [h_{t-1}, x_t] + b_u), \\ r &= \sigma(W_r \cdot [h_{t-1}, x_t] + b_r), \\ \widetilde{h}_t &= \tanh(W_h \cdot [r \cdot h_{t-1}, x_t] + b_h), \\ h_t &= u \cdot h_{t-1} + (1-u) \cdot \widetilde{h}_t. \end{aligned} \tag{3}$$

The output hidden state (h_t) depends on the input and candidate hiden states, in a proportion given by the update gate. The reset gate controls how much the hidden state from the previous step controls the current $\widetilde{h}$. The flow is shown schematically in Fig. 7. Each of the three W matrces is of the size d $\times$ h and the bias vectors b have the size h, where d and h are the dimensions of the input data and hidden state arrays, respectively. The total number of trainable GRU parameters is $3 \cdot$ [h(h+d) + h].

6.2 LSTM

The input to the Long Short-Term Memory (LSTM) cell is the data at the time t (x_t) and the hidden state (h_{t-1}) and the cell memory state (c_{t-1}) from the previous time step. The size of the cell memory state array is the same as the size of the hidden state array. The flow within the cell is controlled by the input

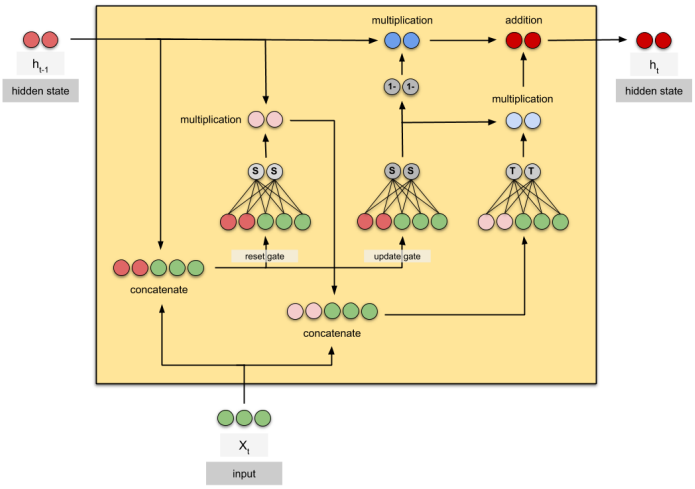

Figure 7. Diagram of a GRU cell with the data input size of 3 and the hidden-state size of 2 (from Ref. [5]).

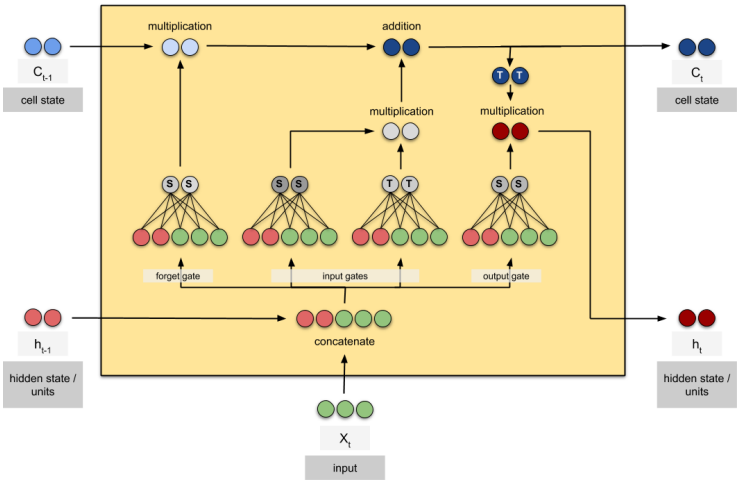

Figure 8. Diagram of an LSTM cell with the data input size of 3 and the hidden-state and memory-cell size of 2 (from Ref. [5]).

gate (i), the forget gate (g), the output gate (o), and the candidate hidden state ($\widetilde{h}$), as given by the following equations:

$$\begin{aligned} i &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \\ f &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \\ o &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \\ \widetilde{h_t} &= \tanh(W_h \cdot [h_{t-1}, x_t] + b_h), \\ c_t &= i \circ \widetilde{h_t} + f \circ c_{t-1}, \\ h_t &= o \cdot \tanh(c_t). \end{aligned} \tag{4}$$

The output hidden state (h_t) depends on the output gate and the cell's current state, where the latter depends on the current hidden state and the previous cell state, in a proportion provided by the input and foget gates. The number of trainable LSTM parameters is $4 \cdot [\mathrm{h(h+d)} + \mathrm{h}]$, where 4 correspond to the number of matrices W in the cell. Fig. 8 shows a schematic diagram of the LSTM cell.

6.3 Bidirectional GRU and LSTM

Bidirectional RNNs are the architectures, in which the standard RNN processing chain is duplicated and the input data is processed in both forward and reverse time order. The two chains are trained with separate parameters and the output states are concatenated. The number of trainable parameters is doubled w.r.t. the standard GRU or LSTM chain.

7. Modeling

The customer next basket content is modeled using an RNN unit (GRU or LSTM) connected with a dense layer (DL) for product classification, using the Keras interface to the Tensorflow library. Two approaches are tested, in which (i) all customers are treated in one model, and (ii) each customer is processed in a separate model. The task is to predict products that a customer already purchased in the past. This condition separates customers into independent product spaces, and hence, the benefits of treating all customers in one model are not immediately clear. This is different, e.g., from the case of a typical recommender system, in which a customer's future preference can be inferred from the preferences of other users. Moreover, the recurrent (time-wise) interaction of a customer with a given product might differ between customers - a feature that may not be captured in one generic model. Therefore, we expect better performance for the separate (per-user) models.

Customer data consists of a 2-dim array, in which the first and the second dimensions run over the customer's orders and products, respecively. The product information is supplied using the one-hot encoding (OHE) technique, for which each column of the array corresponds to a separate product id and contains binary information whether that product is present in a given order. Fig. 9 shows an example of the input data for one of the customers. Only products that are marked as reordered are considered. By definition, this implies that the first order is empty, and we remove it from the dataset. Moreover, a future order will lack products that were bought int he past only once (see the asymptotic value of 0.6 instead of 1 for Recall in Fig. 5, right). We checked that the inclusion of all the products only worsened the model performance, while significantly increased the model dimensionality.

Output probabilities of the DL classifier can be evaluated using the so-called sigmoid or softmax activation functions. The sigmoid provides independent probabilities, which sum up to a value that is typically greater than the unity, while the softmax gives normalized and ordered probabilities that sum up to 1. These characteristics determine the way the future products are predicted (product ranking). In the case of the sigmoid, one estimates a probability threshold above which the products are included in the future order. For the softmax, one selects the first k highest probabilities, assuming the number of products in the future order, k, is know. In our data, the value of k

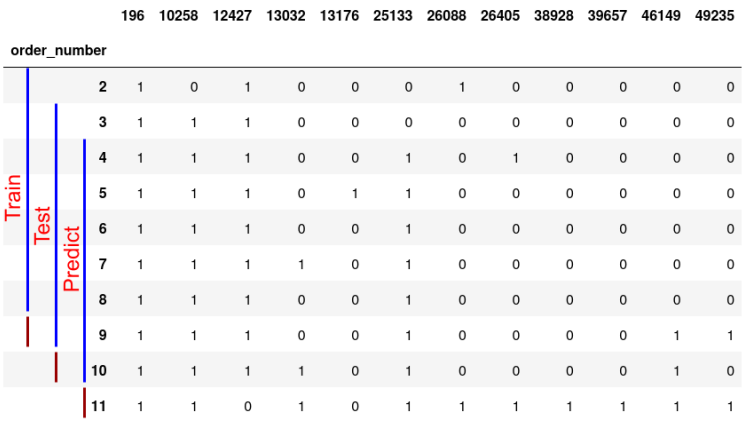

Figure 9. Example of a user's input data to RNN. Columns contain one-hot-encoded products reordered by the user, rows correspond to the user's orders. Colored lines on the left-hand side indicate the orders used for training, testing and model prediction (sliding window method) for per-customer models.

varies significantly from order to order (Fig. 2 bottom) and is not straightforward to predict. Hence, we choose to apply the sigmoid activation function and a threshold-based product ranking, as explained in detail in the next sections.

7.1 Data preprocessing

Technically, the RNN in Keras requires the input data X to be in the format of a 3-dim array, with the second dimension being the recurrent axis. The output data y is returned as a 2-dim array. For per-customer models, the data is formatted as follows:

$$\begin{aligned} X &= [1, \text{orders}_u, \text{OHE}_u], \\ y &= [1, \text{OHE}_u^p], \end{aligned} \tag{5}$$

where orders_u correspond for the user's orders and OHE_u represents the one-hot-encoded list of products present in his/her previous orders (see Fig. 9). To avoid data leakage, the products from the future order are not included in the list. The symbol OHE_u^p denotes product probabilities assigned by the classifier. For the all-customer model, the data has a format:

$$\begin{aligned} X &= [u, \text{orders}_{20}, \text{OHE}_{all}], \\ y &= [u, \text{OHE}_{all}^p], \end{aligned} \tag{6}$$

where the first axis includes now all the users. The same number of orders is required for every user, therefore we truncate them to the last 20 orders or fill the empty rows with zeros. The OHE_{all} is a list of unique products for all the users and all the (20) orders. If no additional selection is performed, this list includes a total of 42,730 products, to be compared to at-most 150 unique products purchased by each user (large sparsity of the combined data). Fig. 10 shows a product frequency plot, constructed as the ratio of total product counts in the sample normalized to the total number of orders (2 million), sorted by

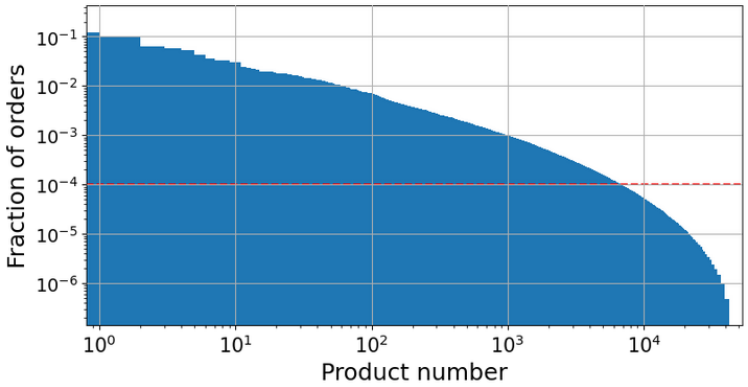

Figure 10. Product freqency plot: total product counts divided by the total number of orders and sorted by decreasing frequency. The red line marks the frequency of 0.1 ‰.

decreasing frequency. The first 20 products are those mentioned in Sec. 3, while the tail of the distribution (>10^4) corresponds to the products present in not more than 100 orders. The red line corresponds to the frequency of 0.1 ‰, and marks the limit of the capacity of our computer. The model performance for various sizes of the product list is studied in more detail in the next section.

Also a separation into the train and test samples depends on the modeling approach. For the all-customer model, a standard method is applied, in which the initial dataset is split into two parallel subsets. The splitting is done in a proportion of 75/25 for the train/test sample, respectively. The advantage of this is that future orders are used in the model training. On the other hand, per-customer models can only use information corresponding to one customer (no parallelization possible), and the sample must be split using the so-called sliding window method. This means that the following ranges of the customer's orders are used for:

- X = [0, orders_u - 3], y = [orders_u - 2] - training,
- X = [1, orders_u - 2], y = [orders_u - 1] - testing,
- X = [2, orders_u - 1], y = [orders_u] - prediction,

as also depicted by the blue and red lines in Fig. 9. A disadvantage of this method is that not only the future order but also the last two previous orders do not contribute to the training phase.

7.2 All-customer model

The performance of the all-customer model is investigated using the GRU+DL architecture for several reduced product lists. The summary of the study is presented in Tab. 1. The number of products in the list (OHE_{all}) is selected by requiring that the product frequency from Fig. 10 be greater than $f > f_{min}$, where f_{min} varies from 3 to 0.1 ‰. The numbers in the brackets correspond to an alternative cut on product counts only (i.e. not divided by the total number of 2M orders). The number of trainable model parameters is evaluated after setting the size of the GRU hidden state to 1/3 of the input data size ($1/3 \cdot \text{OHE}_{all}$),

f_{min} (n_{min})	OHE$_{all}$	n.pars	time	thres.	mean f1
0.003 (6139)	271	123k	7 m	0.18	0.222
0.002 (4094)	450	339k	11 m	0.15	0.242
0.001 (2047)	994	1.6M	47 m	0.13	0.267
0.0004 (818)	2,390	9.5M	4.5 h	0.12	0.288
0.0002 (409)	4,231	29.8M	9 h	0.11	0.287
0.0001 (204)	6,867	78.6M	memory limit		

Table 1. Summary of the GRU+DL performance for different number of customer products in the all-customer model, selected by applyng a cut $f > f_{min}$ (see the text for details). Columns correspond to the number of products, the number of trainable GRU+DL parameters, the model training time, the applied sigmoid probability threshold, and the mean f1 score, respectively. The horizontal lines indicate the best model.

chosen as optimal in a separate study. This number increases very fast with the number of products. Also the performance of the model increases with OHE$_{all}$, until it reaches a plateau or a maximum for f_{min} = 0.4 ‰ (OHE$_{all}$ = 2,390), eventually hitting our computer memory limit at f_{min} = 0.1 ‰. We choose the model corresponding to OHE$_{all}$ = 2,390 as optimal.

Fig. 11 presents the *binary_crossentropy* loss function separately for the train and the test samples as a function of the number of epochs for the optimal model. Adam optimizer, with the stochastic gradient descent and the adjustable learning rate, is used for the model training. The EarlyStopping callback terminates the training if the loss function does not improve after three iterations. The training converges pretty quickly, after only 3 iterations,

Fig. 12 shows the DL classifier's output probabilities for each possible product in each future order (top), together with the threshold optimization scan (bottom). The probability distribution peaks at 0 and exhibits only a small tail toward higher values, which reflects the fact that the data is very sparse. An optimal

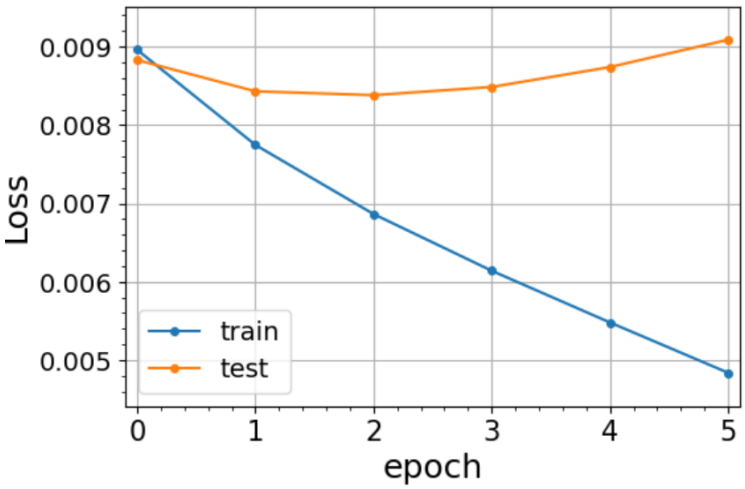

Figure 11. Loss function vs epoch number for the train and test samples (all-customer GRU model).

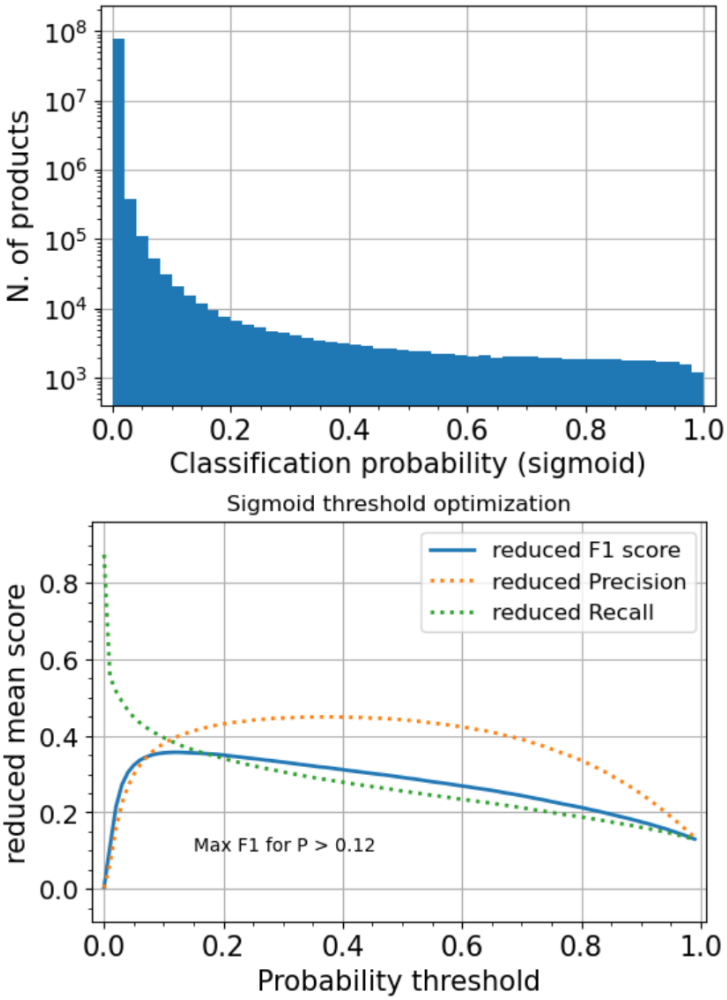

Figure 12. Classifier probabilities (top) and performance scores as a function of probability threshold (bottom) for the all-customer GRU model.

probability threshold of 0.12 is selected in an iterative procedure as the one that maximizes the mean f1 score, calculated using actual products in the future order and predicted products, i.e. those that have probabilities above the threshold. The list of products in the actual future order is reduced to only those that appear in the OHE$_{all}$ list.

The best performing model scores at mean f1 = 0.288, which is significantly below the result of 0.355 obtained with the baseline model, defined as the sum of products in the last two customer's orders.

7.3 Individual-customer models

The performance of individual-customer models is investigated using the RNN+DL architecture with the single- and bi-directional GRU and LSTM cells. To speed up the evaluation process, we build the models only for the first 5,000 customers in the sample. Distinct from the previous case, all the customer's products (OHE_u) are input into the network now. The number of

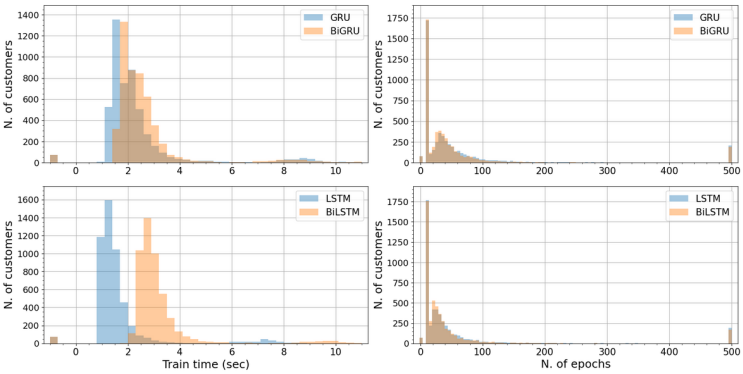

Figure 13. Training time (left) and number of epochs (right) for the GRU-based (top) and LSTM-based (bottom) models.

trainable parameters varies from a few hundred to a few thousand. Fig. 13 shows the model training time and the number of epochs until the model convergence, for each of the RNN types (4 models per customer). Typically, 1 to 4 seconds are needed to train an individual model, with the binary_crossentropy loss function, Adam optimizer, EarlyStopping = 10, and the size of the hidden state equal to the size of the input data OHE_u. The longest time is needed to train the BiLSTM cell. The number of epochs until convergence (<100) is higher than that for the all-customer model, which can be explained by larger fluctuations in the single-user data.

The distribution of the DL classifier output probabilities is presented in Fig. 14, showing no significant excess at 0, which implies that the data is minimally sparse. A prononced excess at 0.5 is typically obtained from the models that are poorly trained. In our data, this excess is larger for the LSTM-based models, which require more trainable parameters, while it is the smallest for the BiGRU model.

Three methods are explored for the product ranking, i.e. to convert DL probabilities into the list of predicted products:

- FUT: the threshold is optimized by comparing products in the actual future order with the predicted future products, i.e. using X = [2, orders$_u$ - 1] and y = [orders$_u$] data introduced in Sec. 7.1, similarly to the technique used for the all-customer model from the previous section. The list of products in the actual future order is reduced to only those that appear in the OHE$_u$ list. This method exploits the user's future data (data leakage) and cannot be applied when models are deployed for production. However, it corresponds to the situation, in which the optimal threshold is estimated with minimal uncertainty, allowing us to evaluate the performance of the RNN part of the model. Fig. 15 shows the distribution of the optimal threshold extracted using this method.
- TES: the threshold is optimized by comparing products in the actual order with the predicted order in the test data, i.e. X = [1, orders$_u$ - 2] and y = [orders$_u$ - 1]. Then, the estimated threshold is applied to the y = [orders$_u$] data. The idea behind this method is that for a well-trained model the probability distributions should generalize well across recurrent orders. This method can be applied after model deployment.
- DTA (Decision-Theoretic Approach, described in Ref. [6]): this method is solely based on predicted probabilities and does not require reference future data. The probabilities are ordered in decreasing order and the predicted product list is truncated at the number of products that maximizes the f1 score that is calculated using an analytical formula. The formula is derived under the assumption that for the optimal prediction the probabilities of being positive for irrelevant items are not more than those for relevant items. The method assumes that product probabilities are independent of each other (a condition that is fulfilled by the sigmoid activation function), and is expected to outperform the previous method for well-trained models and for rear classes.

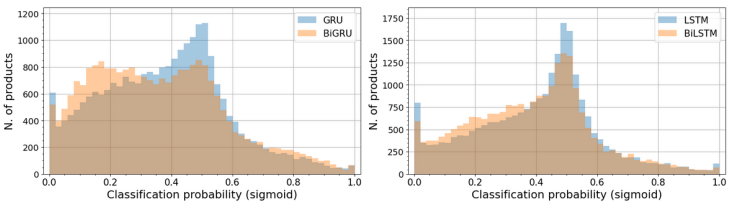

Figure 14. Product probabilities obtained with a classifier using the sigmoid activation function, for the GRU-based (left) and LSTM-based (right) models.

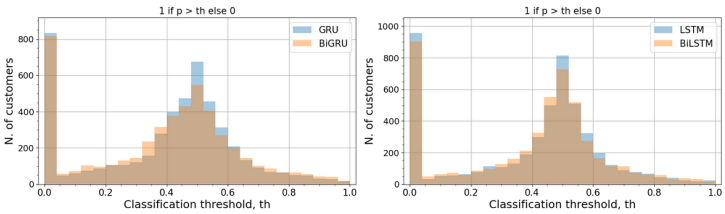

Figure 15. Optimal probability thresholds estimated using the FUT method described in the text, for the GRU-based (left) and LSTM-based (right) models.

Tab. 2 shows the evaluation of the GRU, LSTM, BiGRU, and BiLSTM models, in terms of the mean f1 score, for the three above-mentioned methods of threshold optimization and product ranking. The scores corresponding to the FUT method

Ranking method	GRU	LSTM	BiGRU	BiLSTM
FUT	*0.463*	*0.456*	*0.466*	*0.462*
TES	0.315	0.311	0.319	0.318
DTA	0.321	0.317	0.326	0.321

Table 2. Mean f1 score for the GRU, LSTM, BiGRU and BiLSTM models and three different product ranking methods described in the text, for the first 5,000 customers in the data.

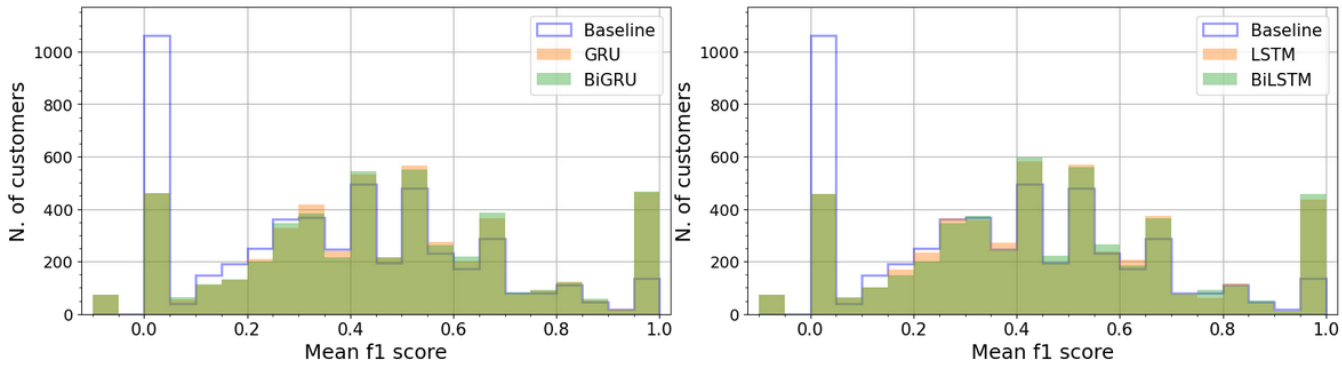

Figure 16. Individual f1 scores for thresholds obtained with the FUT method, for the GRU-based (left) and LSTM-based (right) models, compared to the baseline model.

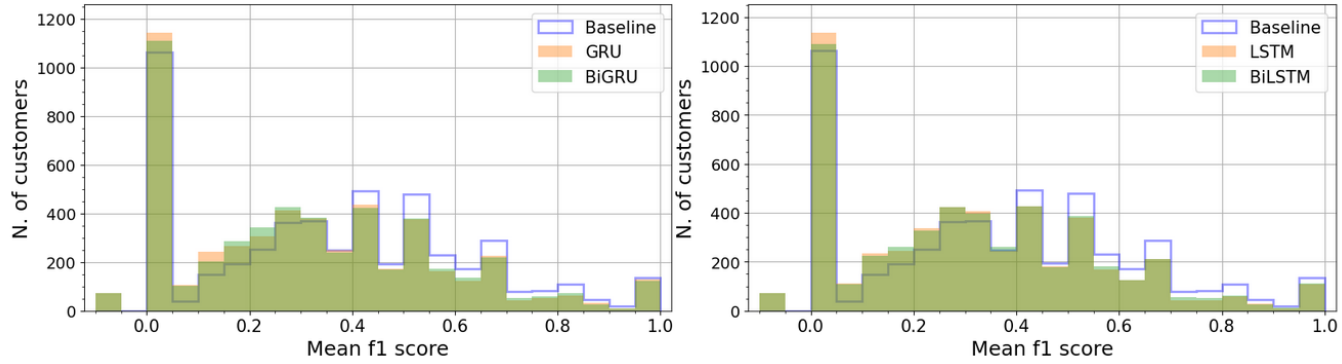

Figure 17. Individual f1 scores for thresholds obtained with the TES method, for the GRU-based (left) and LSTM-based (right) modelsm compared to the baseline model.

are given in italic to mark the fact that the data leakage is present in the FUT prediction. One apparent observation is that the FUT approach significantly outperforms the baseline model, for which the mean f1 score is 0.355. This implies that the RNN part of the model architecture performs remarkably well. However, when the TES and DTA methods are used, the f1 score drops down by a large amount, and the RNN+DL model performance becomes worse than that of the baseline model. We conclude that the threshold optimization is a bottleneck of modeling of the Instacart data. The DTA performs slightly better than the TES method, but more effort needs to be invested in threshold optimization to arrive at the results of the FUT method. For a complete picture, Figs. 16 and 17 show the distribution of the individual customer f1 score for the FUT and TES methods, respectively.

For each of the three product-ranking methods, all the four RNN models have similar performance, although the BiGRU model minimally (but systematically) outperforms other models.

8. Summary

The customer's next basket prediction was modeled using the Instacart data and the RNN+DL neural network architecture, with the RNN being either GRU, LSTM, BiGRU, or BiLSTM unit. We built one model for all the customers, as well as separate models for each customer. The per-customer models have better performance than the all-customer model, reflected in terms of the mean f1 score. However, the RNN+DL models perform worse than the baseline model, define as the copy of the customer's last two orders. The bottleneck of modeling may be attributed to the way a probability threshold is estimated, needed to convert DL classifier probabilities into the future product list.

Future work should focus on improving techniques for threshold estimation, for example by predicting it with a dedicated ML model, such as LightGBM or feed-forward neural network. In addition, the performance of the RNN models could be further imrpoved by adding data features with more information on customers' orders and product buying patterns.

References

[1] https://www.kaggle.com/c/instacart-market-basket-analysis/discussion/38113

[2] https://www.kaggle.com/c/instacart-market-basket-analysis/discussion/38097

[3] https://www.kaggle.com/c/instacart-market-basket-analysis/discussion/38159

[4] https://www.kaggle.com/c/instacart-market-basket-analysis/discussion/33128

[5] https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45

[6] https://arxiv.org/pdf/1206.4625.pdf