Consumer Behaviour in Retail: Next Logical Purchase using Deep Neural Network

Ankur Verma
Indian Institute of Technology BHU
ankur.verma.phe09@iitbhu.ac.in

Abstract

Predicting consumer purchase pattern is one of the most challenging problems for large scale retail firms. Retailers spend a lot of money and resources to ensure smooth and delightful shopping experience. This includes recommending relevant items to consumers and maintaining right inventory. The later helps minimise the instances of Out of stock and Excess Inventory. Consumer purchase prediction problem has generally been addressed by ML researchers in conventional manner either through recommender systems or traditional ML approaches. To our knowledge none of the models have generalized well enough in predicting the consumer purchase pattern. In this paper we present our study of consumer purchase behaviour at the intersection of consumer, item and time using e-commerce retail data. For each of the relevant consumer-item combination, we create a sequential time-series data. We then build generalised non-linear model to predict propensity of consumer to purchase the item for given time horizon. We demonstrate robust performance by experimenting with different neural network architectures, ML models and their combination. We showcase the benefits that neural network architectures like Multi Layer Perceptron, Long Short Term Memory and Convolution Neural Network bring over ML models like Xgboost and RandomForest.

Introduction

Consumer behaviour insights have always been one of the key business drivers for retail, specially given fast changing consumer needs. Existing trend, competitor pricing, item reviews and marketing are some of the key factors driving todays consumer world in retail. While very little information is available on future variablities of the above factors, what retailers have is large volumes of transactional data. Retailers use conventional techniques to model transactional data for predicting consumer choice. While these help in estimating purchase pattern for loyal consumers and high selling items with reasonable accuracy, they don't perform well for the rest. Since multiple parameters interact non-linearly to define consumer purchase pattern, traditional models are not sufficient to achieve high accuracy across thousands to millions of consumers.

In many of the retail brands, short term (4-6 weeks ahead) inventory planning is done on the basis of consumer purchase pattern. Given that every demand planner works on a narrow segment of item portfolio, there is high variability in choices that different planners recommend. Also, given their busy schedule, they have very less interaction moments to discuss their views and insights over their recommendations. Hence, subtle effects like item cannibalization, item affinity, pricing remains unaccounted correctly. Such inefficiencies lead to gap between consumer needs and item availablity, resulting in loss of business opportunities in terms of consumer churn, out of stock and excess inventory.

In this paper, we apply multiple deep learning architectures along with tree based machine learning algorithms to predict the next logical purchase at consumer level. We showcase the performance of individual models with varying hyper configurations. We also show the performance of stacked generalization ensemble and F1-maximization which involves combining predictions from different models and fine tuning purchase probability cut-off at consumer level respectively. The following section explains the overall methodology adopted to solve the problem. It also lays out various algorithmic variants and neural network architectures applied to the problem. Finally, section 3 describes the experiments and results obtained in various scenarios of modelling.

Methodology

We treat each consumer-item as an individual object and generate weekly time series based on historical transaction for each object. The target value at each time step (week) takes a binary input, 1/0 (purchased/not purchased). We adopted walk forward Validation strategy for model testing and generalization. We split the data into 4 parts based on the time series. The last 3 time steps for each of the Time series was used as Validation (hyperparameter Optimization), Test1 (Stacked Generalization) and Test2 (Reporting Accuracy metric) respectively, and the remaining data was used for training. We then generate various types of features including datetime related, label encoded and target encoded within and across objects. Below are the feature groups with the type of features within each group:

- **Datetime:** Transactional metrics at various temporal cuts including week, month and quarter. Fourier and Taylor functions to capture seasonality and trends.

- **Consumer Profile:** Total number of orders by the user, Number of distinct items ordered by the user Time since first order by the shopper, Time since last order by the shopper, Averge time gap between orders Reorder rate of the user, Reorder frequency of the user, Time of the day user visits, Specific item ordered by user in the past Order size based features, Orders with no previously reordered item.
- **Item Profile:** Time since first order for the merchandise, Time since last order for the merchandise Averge time gap between orders, Reorder rate of the item, Reorder frequency of the item Number of co-occuring items with this item, Item's average position in the cart, Statistics around order streak for this item Item's total number of orders, Number of distinct users, Number of users buying it as one shot item.
- **Consumer-Item Profile:** Time since first order for each combination, Time since last order for each combination Averge time gap between orders, Reorder rate of the combination, Reorder frequency of the combination, Streak -user purchased the item in a row, Average position in the cart, Co-occurance Statistics Replacement items, Total number of orders for the combination, If user already ordered the item today.

The model we needed to build, thus, should learn to identify similarly behaving time series across latent parameters, and take into account consumer and item variations in comparing the time series. A row in time series is represented by

$$y_{\text{cit}} = f(i_{\text{t}}, c_{\text{t}}, .., c_{\text{t-n}}, ic_{\text{t}}, .., ic_{\text{t-n}}, d_{\text{t}}, .., d_{\text{t-n}}) \quad (1)$$

where y_{cit} is sales for consumer 'c' item 'i' at time 't'. i_{t} is attribute of the item 'i' like category, department, brand, color, size, etc. c_{t} is attribute of the consumer 'c' like age, sex and transactional attributes. ic_{t} is transactional attributes of the consumer 'c' towards item 'i'. d_{t} is derived from datetime to capture trend and seasonality. Finally, n is the number of time lags.

Accuracy Measure

Since we are solving Binary classification problem, Binary Cross-Entropy/Log Loss [2] seemed most logical loss function for training all our models. Also, the same metric was used during stacking to generate final probabilities for F1-Maximization. But, we used F1-Score [5] values for reporting accuracy as we wanted to maintain a good balance between Recall and Precision.

$$H_{\text{p}} = -\frac{1}{N}\sum_{i=1}^{N} y_{\text{i}}.log(p(y_{\text{i}})) + (1 - y_{\text{i}}).log(1 - p(y_{\text{i}})) \quad (2)$$

where y is the label and p(y) is the predicted probability. Precision refers to the ratio of number of predicted items actually purchased from the predicted items

$$Precision = \frac{TP}{TP+FP} \quad (3)$$

where TP refers to True Positive and FP refers to False Positive. Recall refers to the ratio of predicted items actually purchased from the items in the actual basket.

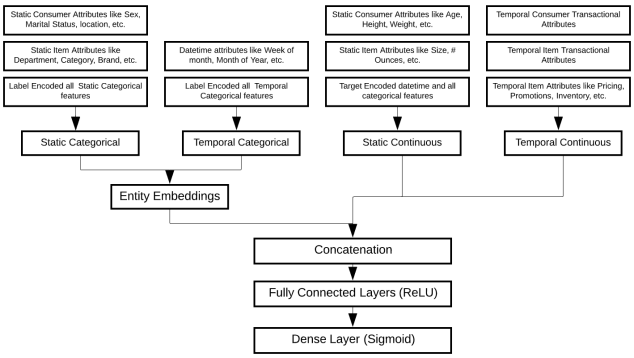

Figure 1: Multi Layer Perceptron (MLP)

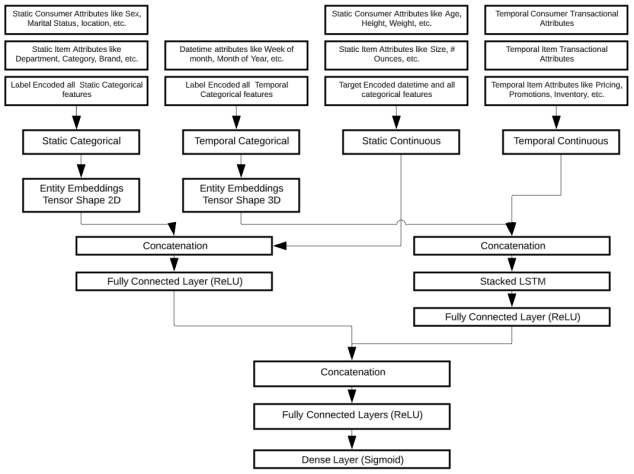

Figure 2: Long Short Term Memory (LSTM)

$$Recall = \frac{TP}{TP+FN} \quad (4)$$

where TP refers to True Positive and FN refers to False Negative. F1 Score is needed when you want to seek a balance between Precision and Recall. We have previously seen that accuracy can be largely contributed by a large number of True Negatives which in most business circumstances, we do not focus on much whereas False Negative and False Positive usually has business costs, thus F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall and there is an uneven class distribution (large number of Actual Negatives).

$$F1 - Score = \frac{2*Precision*Recall}{Precision+Recall} \quad (5)$$

Model Architectures

As mentioned in previous section, traditional machine learning models are not suitable choice for solving Equation 1. Hence, we work with machine learning tree based models like Random Forest Gradient Boosted Trees to Deep learning models ranging from Multi Layer Perceptron (MLP), Long Short Term Memory (LSTM) and Convolution Neural Network (CNN). Architectures of MLP, LSTM, CONV1D

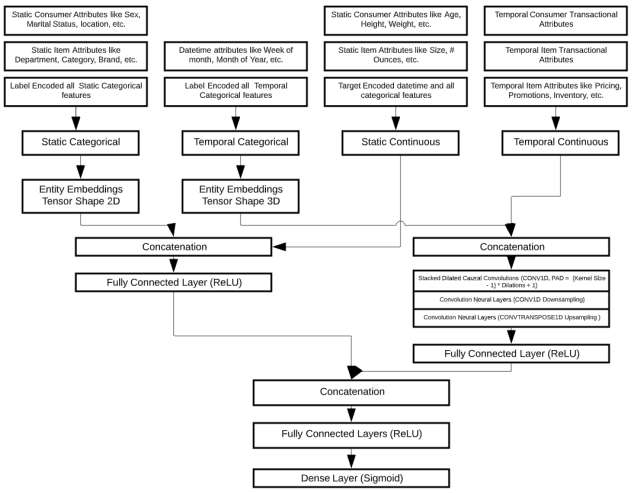

Figure 3: Convolution Neural Network 1D (CONV1D)

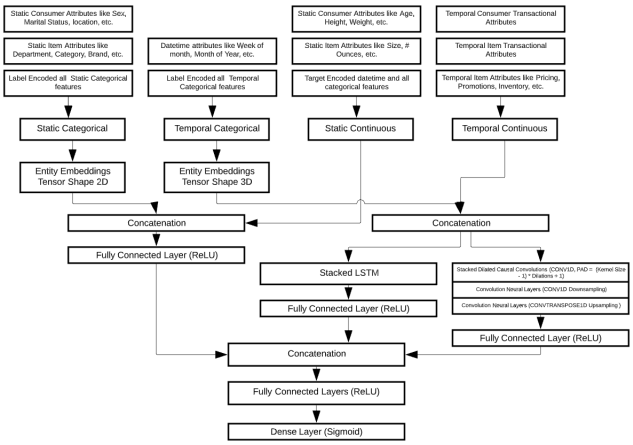

Figure 4: Convolution Neural Network 1D + Long Short Term Memory (CNNLSTM)

and CNNLSTM models are shown in Figure 1, Figure 2, Figure 3 and Figure 4.

The consumer purchase pattern has huge variation in terms of time of purchase (weekday/weekends), cadence of purchase (days to months), purchased item types (dairy/meat/grocery/apparels/etc.) and brand loyalty (tendency to substitute items). Given such huge variance it becomes imperative to cross learn consumer behaviour from like consumer groups. To learn such relationships its very important to capture non-linear relationship between target and regressors at the most granular level. Tree based and Deep learning models are chosen for their ability to model feature interactions even if transient in time, so that they capture non-linearity well. Utility of traditional machine learning algorithms like Logistic regression , SVM and recommender systems are limited given the scale of large retail firms (Millions of customers with thousands of products). Such models do not scale well for large sets of data and hyperparameters.

Tree based models and MLP are trained in such a way where lagged values of time varying features are used to capture temporal dependencies. We use lagged values of temporal features up to last n time steps (n goes till 52 weeks). Multiple Lagged values as well as Statistical rolling operations like mean, median, quantiles, variance, kurtosis and skewness over varying lag periods are used for feature generation. Details around datasets and derived features are explained in the following section. This was decided after some preliminary experiments. Hyper-parameters of tree based models are optimized using Bayesian Hyper-parameter Optimization Technique. LSTM, CONV1D and CNNLSTM models were trained in sequence to sequence fashion using entire life-cycle data of a time series (Consumer-Item).

We applied empirical approach to tune the hyperparameters for Deep learning models. All hyperparameter Optimization was performed over Validation dataset. We list some of the params along with the values we used for Deep learning models.

- **Optimizer Parameters:** RMSProp and Adam were used as different trial configurations. The learning rate was experimentally tuned to 1e-3. We also did weight decay of 1e-5 which helped a bit in model Regularization.
- **Scheduler Parameters:** Cyclic and ReduceLROnPlateau Learning rates were used as different trial configurations. we used 1e-3 as max lr and 1e-6 as base lr for Cyclical learning rate along with the step size being the function of length of train loader. ReduceLROnPlateau was tuned for 1e-6 as min lr.
- **SWA:** Stochastic Weight Averaging (SWA) is used to improve generalization across Deep Learning models. SWA performs an equal average of the weights traversed by SGD with a modified learning rate schedule. We used 1e-3 as SWA learning rate.
- **Regularization Parameters:** We also L1 and L2 norm for better generalisation of the models.

Apart from the above parameters we also iterated enough to tune network parameters like number of epochs, batch size, number of Fully Connected Layers, number of LSTM layers, convnet parameters (kernel size, dilations, padding) and embedding sizes for the categorical features. BCELOSS was used a loss function for the backward and forward passes in Deep Learning setting. Deep learning models are built using deep learning framework PyTorch, and are trained on GCP instance containing 6 CPUs and a single GPU. scikit-learn is used for Tree based models like RandomForest and XGBoost. We built a total of 46 models, 9 different param configurations for each of 4 Deep Learning models and 5 best trials for each of 2 Machine Learning models as shown in table 1.

Stacked Generalization

For simplicity and better generalization we adopted non-parameteric approach for stacking. We used Weighted K Best Stacked Generalization Ensemble for combining the 46 models (candidates) from Table 1. Test1 Logloss was used

Table 1: Model Types with Specification

Model Type	Number of Selected Trials	Model Tuning parameters
MLP	9	Optimizer, Scheduler, SWA, Parameter Averaging, Feature Groups, FC Layers
LSTM	9	Optimizer, Scheduler, SWA, Parameter Averaging, Feature Groups, FC Layers, LSTM Layers
CONV1D	9	Optimizer, Scheduler, SWA, Parameter Averaging, Feature Groups, FC Layers, Convolution Parameters
CNNLSTM	9	Optimizer, Scheduler, SWA, Parameter Averaging, Feature Groups, FC Layers, LSTM, Convolution Parameters
Xgboost	5	Learning rate, Tree Depth, Regularization parameters
RandomForest	5	Tree Depth, Regularization parameters

as metric to compute weight for each of the 46 candidates. We set the k to 25 and took the Weighted Average of the top 25 candidates based upon Test1 Logloss.

F1-Maximization

Apart from the above parameters we also iterated enough to tune network parameters like number of epochs, batch size, number of Fully Connected Layers, number of LSTM layers, convnet parameters (kernel size, dilations, padding) and embedding sizes for the categorical features.Deep learning models are built using deep learning framework PyTorch, and are trained on GCP instance containing 6 CPUs and a single GPU. scikit-learn is used for Tree based models like RandomForest and Xgboost.