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

Ankur Verma
Deep Learning Engineer
ankur.verma.phe09@iitbhu.ac.in

Abstract

Predicting future consumer behaviour is one of the most challenging problems for large scale retail firms. Accurate prediction of consumer purchase pattern enables better inventory planning and efficient personalized marketing strategies. A good inventory planning helps minimise the instances of Out of stock and Excess Inventory as well as smart personalized marketing ensures smooth and delightful shopping experience. 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 traditional models have not generalized well enough in predicting the consumer purchase pattern. In this paper, we present our study of consumer purchase behaviour wherein we establish a data-driven framework to predict whether a consumer is going to purchase an item within a certain time frame using e-commerce retail data. To model this relationship, we create a sequential time-series data for each of the relevant consumer-item combination. We then build generalised non-linear models by generating features at the intersection of consumer, item and time. We demonstrate robust performance by experimenting with different neural network architectures, ML models and their combination. We showcase the benefits neural network architectures like Multi Layer Perceptron, Long Short Term Memory and Temporal Convolutional Networks bring over ML models like Xgboost [18] and RandomForest.

1 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.

2 Related Work

In the last decade, various machine learning methods for predicting consumer purchase pattern have been analyzed in the academia field and some of them are often used by practitioners. In most cases those approaches are based on extracting customer's latent characteristics from its past purchase behavior and applying ML formulation. Previous studies have analyzed the use of random forest and Xgboost techniques in order to predict customer retention. Past consumer behavior was used as potential explanatory variable for modelling the purchase pattern.

Closely related to our work is, where the authors develop a model for predicting wether a customer performs a purchase in some prescribed future time frame based on purchase information from the past. They propose customer character-

Table 1: Modelling data splits

Data Split	Specifications	Consumer-Item combinations	Max Time-Series length
Train	Model training	5,872	46 weeks
Validation	Hyper-Parameter Optimization	5,888	2 weeks
Test1	Stacked Generalization, F_1-Maximization	5,899	2 weeks
Test2	Reporting Accuracy Metrics	5,910	2 weeks

istics such as the number of transactions observed in past time frames, time of the last transaction, and the relative change in total spending of a customer. They found an gradient tree boosting to perform best on the tested data. We propose neural network architectures which outperforms the gradient boosting type of models.

3 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 as shown in Table 1. We then generate various types of features including datetime related, label encoded and target encoded within and across objects. Below are the feature groups we experimented with explicitly or unexplicitly:

- **Datetime:** Transactional metrics at various temporal cuts including week, month and quarter. Datetime related features capturing seasonality and trend.
- **Consumer-Item Profile:** Transactional metrics at different granularities including consumer, item, consumer-item, department, aisle, etc. The metrics includes some of likes Time since first order, Time since last order, time gap between orders, Reorder rates, Reorder frequency, Streak - user purchased the item in a row, Average position in the cart, Total number of orders.
- **Consumer-Item-Time Profile:** Transactional metrics at the intersection of consumer, item and time. Interactions capturing consumer behaviour towards items for the given time period.

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

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

where y_{cit} is purchase prediction 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.

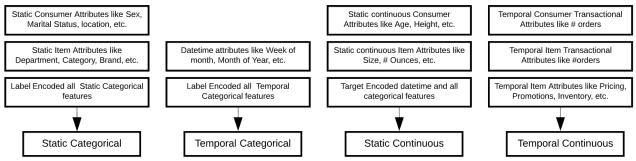
3.1 Loss Function

Since we are solving Binary classification problem, Binary Cross-Entropy/Log Loss seemed most logical loss function for training all our models.

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

where H_p is total loss and y_i is the label and $p(y_i)$ is the predicted probability.

Figure 1: Data classification for DNN Architectures



3.2 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 Temporal Convolutional Network (TCN). Architectures of MLP, LSTM, TCN and TCN-LSTM models are shown in Figure 2, Figure 3, Figure 4 and Figure 5. As described in architecture diagrams Figure 1, data was classified into following groups:

- **Static Categorical:** These are categorical features which donot vary with time. This includes the consumer attributes like sex, marital status and location, different item attributes like category, department and brand.
- **Temporal Categorical:** These are categorical features which vary with time. It includes all the datetime related features like week, month of year, etc.
- **Static Continuous:** These features are static but Continuous. This includes certain consumer attributes like age and weight, item attributes like size and certain derived features like target encoded features.
- **Temporal Continuous:** These are time varying Continuous features. All consumer and item related traditional attributes like number of orders, add to cart order, etc. falls under this class.

In all the above neural network architectures, we learn the embeddings of the categorical features during training phase. We embed these attributes in order to both compress their representations while preserving salient features, as well as capture mutual similarities and differences. 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

Figure 2: Multi Layer Perceptron (MLP)

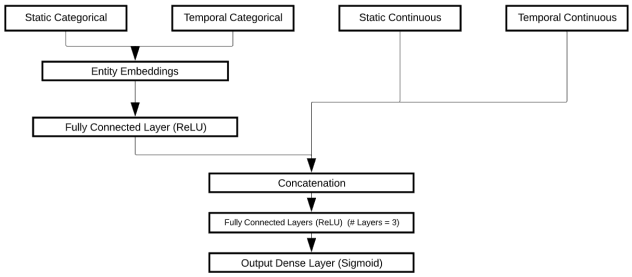
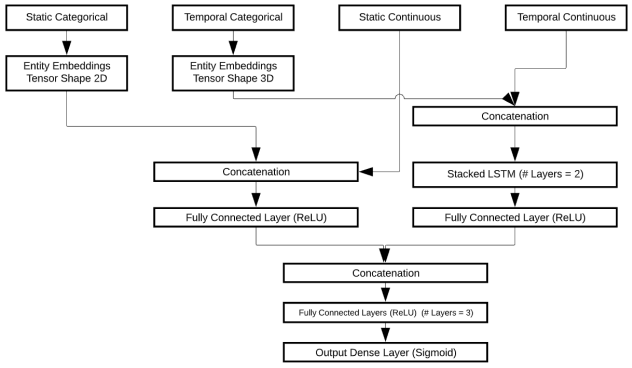


Figure 3: Long Short Term Memory (LSTM)



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 Hyperparameter Optimization Technique. LSTM, TCN and TCN-LSTM 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.
- **Parameter Average:** This applies weighted average over the parameter weights of n best performing epochs (local minimas) from state dictionary post completion of training.

Apart from the above parameters we also iterated enough to

Figure 4: Temporal Convolutional Network (TCN)

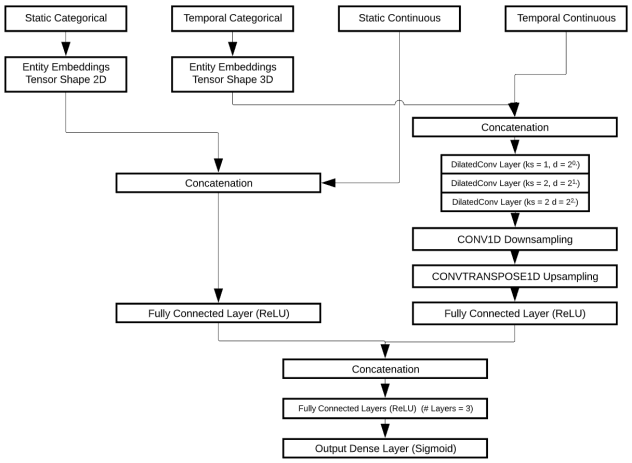
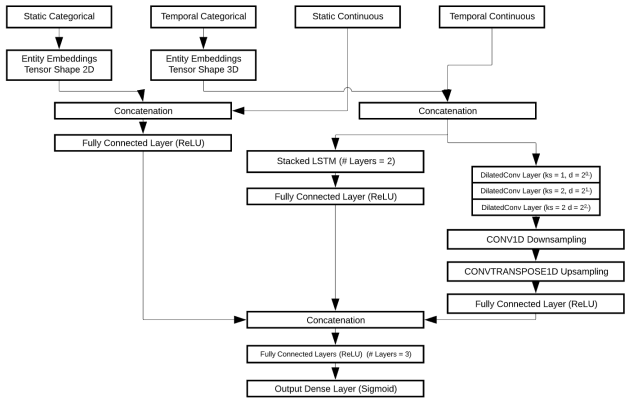


Figure 5: TCN-LSTM



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. Binary Cross-Entropy/Log Loss 2 was used as loss function for all the models. We used Bayesian Optimization Technique for hyper configuration tuning with 100 trials. Some of the hyper-parameters we used for Machine learning models are :

- **Learning Rate:** LR range was set to vary between 1e-3 to 0.5.
- **Max Depth:** Depth range was set between 2 to 12 at the step of 1.

Apart from these, Regularization parameters like Reg Lambda, Min Sample Leaf was also tuned from Bayesian framework.

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 60 models, 12 different param configurations for each of 4 Deep Learning models and 6 best trials for each of 2 Machine Learning models as shown in table 1.

3.3 Stacked Generalization

For simplicity and better generalization we adopted non-parameteric approach for stacking. We used Weighted K-Best and K-Best as Stacked Generalization Ensemble model for combining the 60 models (candidates) from Table 1. Test1 BCELoss was used as metric to compute weight for each of the 60 candidates. We set k to 5, 10 and 25 and selected these top candidates based upon the experiments performed over test1 metrics. Finally, we took Weighted Average of the probabilities for Train, Validation, Test1 and Test2 time steps for each consumer-item combination.

3.4 F_1-Maximization

Post generation of the consumer-item probabilities, we optimize for the purchase cut-off probability based on probability distribution of test1 at consumer level. For instance, lets say we generated purchase probabilities for n_i items of b_i actually purchased items for consumer c_i. Let Actual items purchased by consumer c_i at test1 time step be [Ia_1, Ia_2 ,.., Ia_{b_i}] whereas items for which the model generated probabilities for the consumer c_i at test1 time step be [Ip_1, Ip_2 ,.., Ip_{n_i}].

$$A_{c_i} = [a_1, a_2, .., a_{n_i}] \; \forall \; a_j \in \{0,1\} \tag{3}$$

$$P_{c_i} = [p_1, p_2, .., p_{n_i}] \; \forall \; p_j \in \; [0, 1] \tag{4}$$

A_{c_i} represents the actuals for consumer c_i, with a_j being 1/0 (purchased/non purchased). P_{c_i} represents the predicted probabilities for consumer c_i for respective item, with p_j being probability value. As mentioned above n_i is the total items model generated purchase probabilities for.

$$D(Pr_{c_i}) : {P_{c_i}}^{1 \times n_i} \rightarrow {P'_{c_i}}^{1 \times n_i} \quad : p'_j = \begin{cases} 1 & p_j \geq Pr_{c_i} \\ 0 & p_j < Pr_{c_i} \end{cases} \tag{5}$$

$$P'_{c_i} = [p'_1, p'_2, .., p'_{n_i}] \; \forall \; p'_j \in \{0,1\} \tag{6}$$

$$k_i = \sum_{i=1}^{n_i} p'_i \tag{7}$$

Pr_{c_i} is the probability cut-off. Decision rule D converts probabilities P_{c_i} to binary predictions P'_{c_i} such that if p_j is less than Pr_{c_i} then p'_j equals 0 else 1. k_i is the total number of predictions which Decision rule D converted to 1 or in other words number of predictions which model predicted as purchase.

$$V_{Pr_{c_i}} = P'_{c_i} \times {A_{c_i}}^T \Rightarrow \begin{pmatrix} p'_1 & .. & p'_{n_i} \end{pmatrix} \times \begin{pmatrix} a_1 \\ .. \\ a_{n_i} \end{pmatrix} \tag{8}$$

$V_{Pr_{c_i}}$ represents the number of items with purchase probabilities greater than Pr_{c_i} which were actually purchased. Now using the below formulae we calculate Precision, Recall and F_1-score for consumer c_i.

$$Precision_{c_i} = \frac{V_{Pr_{c_i}}}{k_i} \quad and \quad Recall_{c_i} = \frac{V_{Pr_{c_i}}}{b_i} \tag{9}$$

$$F_{1_{c_i}} = \frac{2 \times Precision_{c_i} \times Recall_{c_i}}{Precision_{c_i} + Recall_{c_i}} \quad \Rightarrow \quad 2 * \frac{V_{Pr_{c_i}}}{k_i + b_i} \tag{10}$$

$F_{1_{c_i}}$ becomes the value to be maximised by finding optimal Pr_{c_i} for consumer c_i. We apply the above Optimization function over test1 probability distribution at a consumer level to find optimal cut-off. Final purchase predictions happens based on the cut-off value.

4 Experiments and Results

We use transactional data from instacart kaggle challenge to train all our models. As can be seen in Figure 6 data has transactional details including consumer id, item id, order id, add to cart order, date of transaction, aisle id and department id. Also, from Table 1, we can see that we utilize 1 year data which gets split into train, validation, test1 and test2. We generate consumer-item-week level data with purchase/ non purchase being the target. We use the above data to generate consumer-item purchase predictions for forwards 2 time steps (2 weeks for our case).

4.1 Experiment Setups

We started with exploratary data analysis, looking at the data from various cuts and trying to study the variations of the features with target. Some of them include looking at the density of consumers with different basket sizes Figure 8, orders placed for items across departments Figure 7, variation of order probability with add to cart order Figure 9, order probability variations at different temporal cuts like week, month and quarter, transactional metrics at both consumer and item levels like total orders, total reorders, recency, gap between orders, etc. We then performed multiple experiments with above features and different hyper-configurations.

Table 2: Model Specifications

Model Type	Trials	Model Hyper-Parameters	Loss Functions
MLP	12	Optimizer, Scheduler, SWA, Parameter Averaging, Feature Groups, FC Layers	BCELoss
LSTM	12	Optimizer, Scheduler, SWA, Parameter Averaging, Feature Groups, FC Layers, LSTM Layers	BCELoss
TCN	12	Optimizer, Scheduler, SWA, Parameter Averaging, Feature Groups, FC Layers, Convolution Parameters	BCELoss
TCN-LSTM	12	Optimizer, Scheduler, SWA, Parameter Averaging, Feature Groups, FC Layers, LSTM, Convolution Parameters	BCELoss
Xgboost	6	Learning rate, Tree Depth, Regularization parameters	BCELoss
RandomForest	6	Tree Depth, Evaluation Metrics, Regularization parameters	BCELoss

Figure 6: Sample Dataset

user_id	order_id	product_id	add_to_cart_order	order_number	date	product_name	aisle_id	department_id	aisle	department
1	2539329	196	1	1	01/01/18	Soda	77	7	soft drinks	beverages
1	2539329	14084	2	1	01/01/18	Organic Unsweetened Vanilla Almond Milk	91	16	soy lactosefree	dairy eggs
1	2539329	12427	3	1	01/01/18	Original Beef Jerky	23	19	popcorn jerky	snacks
1	2539329	26088	4	1	01/01/18	Aged White Cheddar Popcorn	23	19	popcorn jerky	snacks
1	2539329	26405	5	1	01/01/18	XL Pick-A-Size Paper Towel Rolls	54	17	paper goods	household
1	2398795	196	1	2	16/01/18	Soda	77	7	soft drinks	beverages
1	2398795	10258	2	2	16/01/18	Pistachios	117	19	nuts seeds dried fruit	snacks
1	2398795	12427	3	2	16/01/18	Original Beef Jerky	23	19	popcorn jerky	snacks
1	2398795	13176	4	2	16/01/18	Bag of Organic Bananas	24	4	fresh fruits	produce
1	2398795	26088	5	2	16/01/18	Aged White Cheddar Popcorn	23	19	popcorn jerky	snacks
1	2398795	13032	6	2	16/01/18	Cinnamon Toast Crunch	121	14	cereal	breakfast

Table 3: BCELoss of Test2 for 12 Trials of Deep Learning Models

Trial	Optimizer	Scheduler	SWA	Parameter Avg	MLP	LSTM	TCN	TCN-LSTM
1	RMSprop	ReduceLROnPlateau	True	False	**0.0276**	0.0306	**0.0249**	0.0307
2	RMSprop	CyclicLR	True	False	0.0708	**0.0269**	**0.0269**	0.0348
3	Adam	ReduceLROnPlateau	True	False	0.0295	0.0303	0.0667	0.0337
4	RMSprop	ReduceLROnPlateau	False	False	0.2427	**0.0275**	0.0364	0.0759
5	RMSprop	CyclicLR	False	False	**0.0250**	0.0306	0.0600	**0.0286**
6	Adam	ReduceLROnPlateau	False	False	0.0360	**0.0302**	0.0590	0.0309
7	RMSprop	ReduceLROnPlateau	False	True	0.2903	0.0432	0.0453	0.0381
8	RMSprop	CyclicLR	False	True	**0.0245**	0.0378	0.0569	**0.0262**
9	Adam	ReduceLROnPlateau	False	True	0.0700	0.0491	0.0610	0.0382
10	RMSprop	ReduceLROnPlateau	True	True	0.0356	0.0364	**0.0238**	0.0309
11	RMSprop	CyclicLR	True	True	0.0420	0.0377	0.0284	**0.0269**
12	Adam	ReduceLROnPlateau	True	True	0.0321	0.0306	0.0547	0.0305

Table 4: BCELoss of Test2 for 6 best Trials of ML Models

Trial	Hyper-Parameter	Xgboost	RandomForest
1	Bayesian-Optimizer	**0.0332**	0.0526
2	Bayesian-Optimizer	0.0364	0.0479
3	Bayesian-Optimizer	0.0347	**0.0416**
4	Bayesian-Optimizer	0.0364	**0.0449**
5	Bayesian-Optimizer	**0.0335**	**0.0459**
6	Bayesian-Optimizer	**0.0339**	0.0578

Table 5: BCELoss mean of top 3 trials across data splits

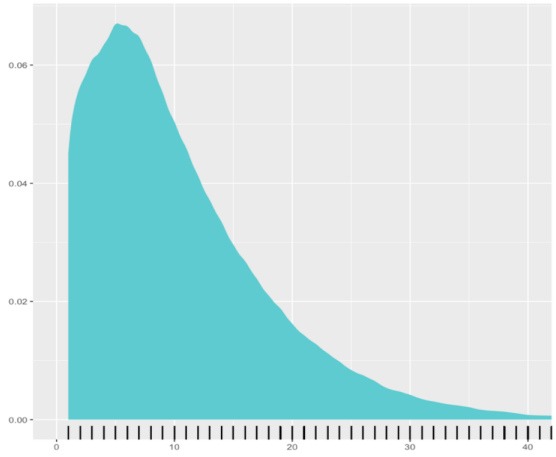
Model Type	Val BCELoss	Test1 BCELoss	Test2 BCELoss
MLP	0.0405	0.0289	0.0256
LSTM	0.0373	0.0293	0.0282
TCN	**0.0368**	**0.0292**	**0.0251**
TCNLSTM	0.0368	0.0304	0.0273
Xgboost	0.0352	0.0318	0.0335
RandomForest	0.0437	0.0389	0.0441

4.2 Results and Observations

Tables 3 and 4 shows the experimental results obtained with different hyper-configurations over multiple models. From Table 3, we can infer that RMSprop and CyclicLR emerged as the clear winners as Optimizer and Scheduler respectively. TCN showed the best Accuracy score at test2, with most of the other Deep Learning models have comparable results. Also, we observe that all the Deep Learning models out perform Machine Learning models including Xgboost and RandomForest both in terms of Accuracy and Generalization. This insight can be drawn from Table 5 which has the average of BCELoss across 3 best trials. Range of scores across val , test1 and test2 states that Deep Learning models are showing more stable results in terms of model fit. We present the effectiveness of combining predictions in the form of stacking. From Table 6, we can see the results of stacking at different values of K, both for Weighted K-

Figure 7: Most ordered Items across Departments

Figure 8: Density of consumers Vs. Basket Size



Maximum number of consumers have their basket size between 5-7

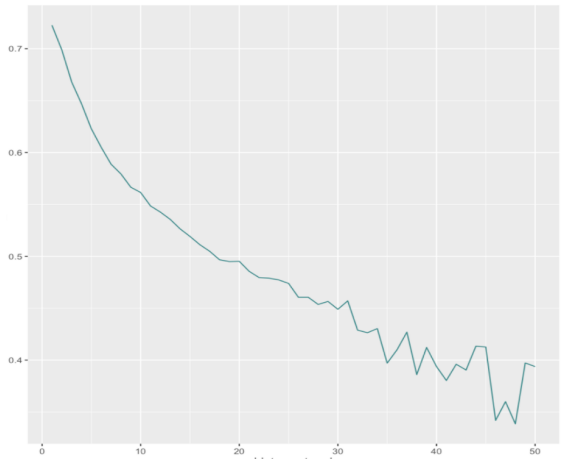
Best as well as K-Best model setups. Finally we apply F_1-Maximization so as to be able to strike a balance between Precision and Recall. After F_1-Maximization we see that all Precision, Recall and F_1-Score are close enough for all data splits as can be seen from Table 7.

4.3 Industrial Applications

The Next Logical Purchase framework has multiple applications in retail industry. Some of its applications include

- **Personalized Marketing:** Item Recommendations at consumer level can be derived from the solution of this problem, which will enhance consumer shopping experience.
- **Inventory Planning:** This can help in Planning inventory well, as it provides the consumer choice at a point time.

Figure 9: Reorder probability Vs. Add to cart order



Probability of reordering decreases as we go higher up in add to cart order

Table 6: Stacked Generalization Results

Model Type	K Value	Val BCELoss	Test1 BCELoss	Test2 BCELoss
Weighted K Best	**3**	**0.0386**	**0.0278**	**0.0242**
Weighted K Best	5	0.0373	0.0282	0.0245
Weighted K Best	10	0.0397	0.0290	0.0258
Weighted K Best	15	0.0389	0.0296	0.0272
Weighted K Best	25	0.0394	0.0316	0.0287

- **Assortment Planning:** In retail stores, this can be used for placing items in MOD based on the consumer preference model at a given time frame.

5 Conclusion

We have presented the study of the Consumer behaviour with Deep Neural Networks and shown how DNNs outperform the Machine Learning models like Xgboost and RandomForest. We also showcase the potential of stacked generalisation as well as F_1-Maximization.

References

[1] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

[2] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

[3] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

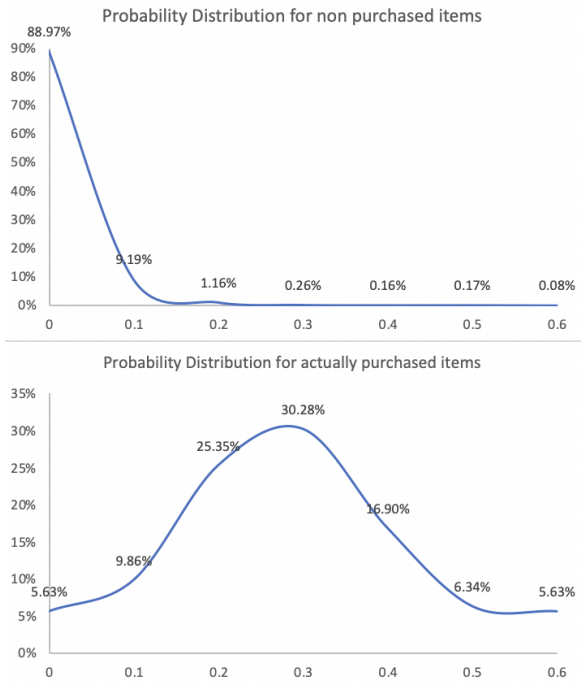
[4] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

[5] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

[6] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

[7] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

Figure 10: Probability Distributions



Probability Distributions for the non purchased cases tends towards 0 and tapers as the probability moves towards 0.5, whereas for purchased cases the distribution witnesses a peak between 0.2 and 0.3 purchased cases 0.5

Table 7: Final Accuracy post F_1-Maximization

Data Split	Precision	Recall	F_1-Score
Validation	0.3401	0.4981	0.4042
Test1	0.3323	0.5103	0.4024
Test2	0.3506	0.4964	0.4109

[8] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

[9] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

[10] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

[11] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

[12] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

[13] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

[14] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

[15] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

[16] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

[17] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.

[18] Tianqi Chen XGBoost: A Scalable Tree Boosting System https://dl.acm.org/doi/10.1145/2939672.2939785.