Project Report

On

Recommender Systems based on click events

Abhinav Anand (2018201037)

Abhijeet Panda (2018201044)

Shubham Rawat(2018201098)

Nishant Goyal (2018201038)

INTRODUCTION

User clicks represent a rich source of information, in which user preferences, behavior patterns, and search intentions are implicitly embedded. It has been well recognized that such information is highly valuable particularly for online retailers to better understand their customers and make corresponding advertising and recommendations so as to increase revenue. Two out of many important questions with respect to user clicks and click sessions are whether a certain click session will result in a buying event, and which items will be bought then.

What are the different types of recommendations?

There are basically three important types of recommendation engines:

- Collaborative filtering
- Content-Based Filtering
- Hybrid Recommendation Systems

Here our main focus predicting the buys which is similar to content-based filtering in which the people clicks data helps us to find the item the user wants to buy.

OVERVIEW

PROBLEM STATEMENT:

The goal is hence to predict whether the user (a session) is going to buy something or not, and if he is buying, what would be the items he is going to buy. Such an information is of high value to an e-business as it can indicate not only what items to suggest to the user but also how it can encourage the user to become a buyer.

We started with an observation that the target objective is not smooth and cannot be optimized directly with any Machine learning model. We also noted that this objective is maximized when all sessions with buy events and all bought items are predicted correctly. This suggested an indirect optimization approach where we first develop session and item models, and then use these models to select subsets of session and items that maximizes our objective.
Our approach is to solve a simpler analogous problem where given an item and its session, the aim is to predict the probability of purchase for the given item. For each session, the predictions can result in a set of purchased items or an empty set meaning there are no purchases during the session. For each item in a session, we engineer features regarding the session, the item properties, and the time window in which it appears. Since the dataset is highly imbalanced, a sampling process is applied to balance class priors either as a preprocessing

step or by embedding it into the classifier logic. This results in a balanced random forest classifier which is trained to perform predictions on the test set.

APPROACH:

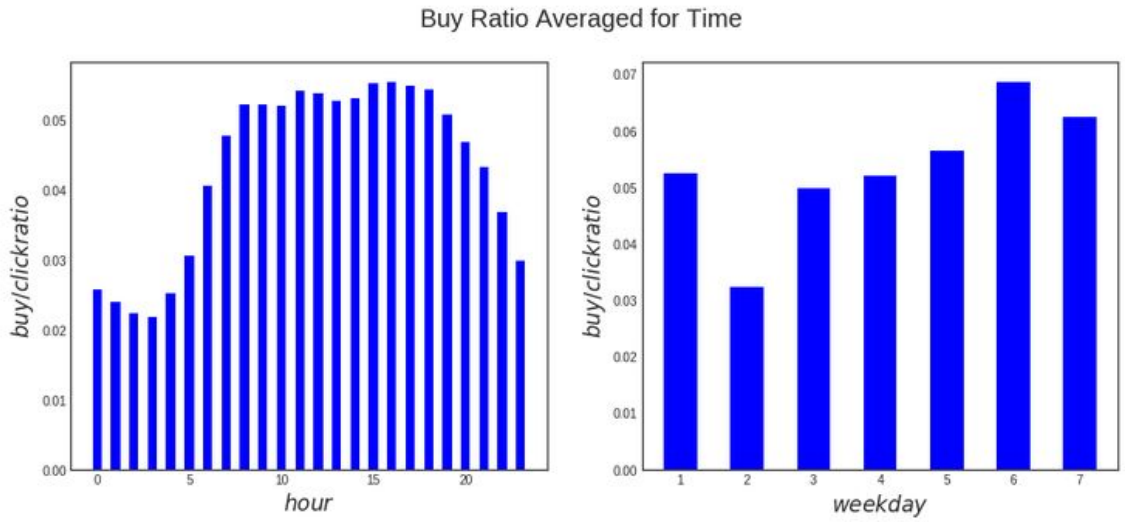
DATA DESCRIPTION:

The data is from YOOCHOOSE containing training and test data files. The training data comprises two different files: yoochoose-clicks.dat for click events and yoochoose-buys.dat for buy events. The test data contains one file: yoochoose-test.dat, which is identically

structured as the yoochoose-clicks.dat of the training data. The first training dataset has 33,003,944 unique clicks from 9,249,729 sessions over 52,739 unique items spanning from 03/31/2014 to 09/29/2014. The other training dataset has all the sessions that are known to have a buying event, referred to as buying sessions. There are 509,696 buying sessions with 1,150,753 buying events over 19,949 items. All such sessions are included in the first training dataset. The sessions without buying events are referred to as no-buying sessions. The test dataset has a disjoint set of sessions without buying information, on which the final submission and evaluation will be. There are 2,312,432 sessions with 8,251,791 clicks over 42,155 items in the test dataset. There are 1,548 new items that only appear in the test dataset but not in the training dataset. In total, there are m = 11, 562, 161 sessions and n = 54, 287 items given in the Challenge.

Now we have a large distribution of data , but that data is too ambiguous, and we had gained a description about how much buys per clicks in our data. Here is the ratio showed in the figure.

First figure says buys/clicks ratio in an hour and second figure is of buys/clicks ratio in weekdays.



By these figures we have seen that our data is too ambiguous. Because the buy/clicks ratio is too small. This shows us that clicks data is too large as compared to buys. So our next step is to extract some features from the data , which help us to do predict in our buys.

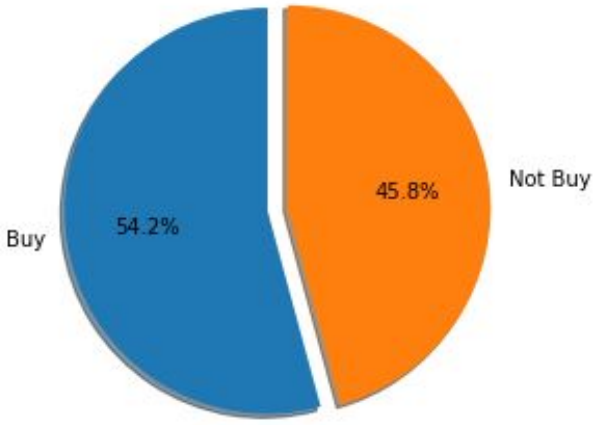
FEATURE EXTRACTION:

Challenge training and test sets comprise tuples in the form of session ID, timestamp, item ID, category . The training set has a corresponding purchase dataset with its tuples session ID, timestamp, item ID, price, quantity showing a purchased item in a session. The data has some missing values for price and quantity as well as item categories. An item can belong to more than one category and the category names are not provided. It is unclear when a session ends after the last item click. Furthermore, some purchase events occur in between item clicks, while most of them occur after the clicks finish.

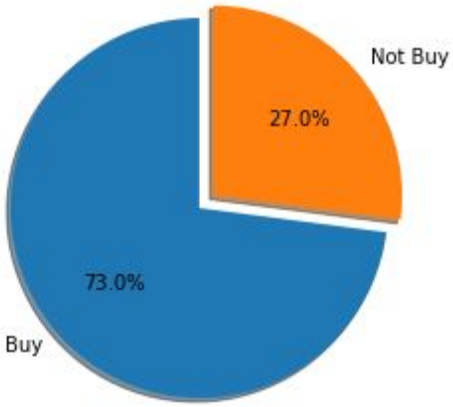
Table of the exploitary analysis:

What happened when an item	**Buy**	**Not Buy**
Clicked maximum time in session	72.96	27.04
Clicked minimum number of time	54.21	45.78
Maximum time spent on an item in a session	61.75	38.24
Minimum time spent on an item in a session	54.23	45.76

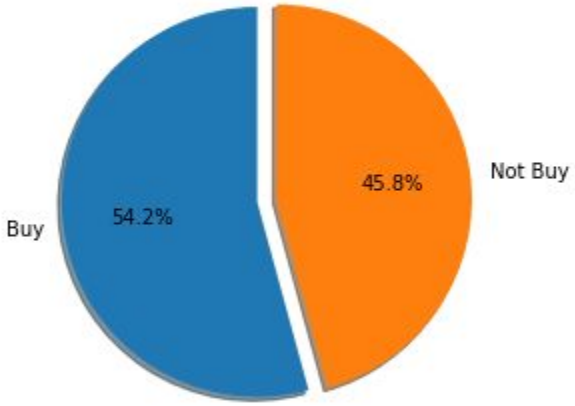
Items Bought when clicked minimum number of times in a session



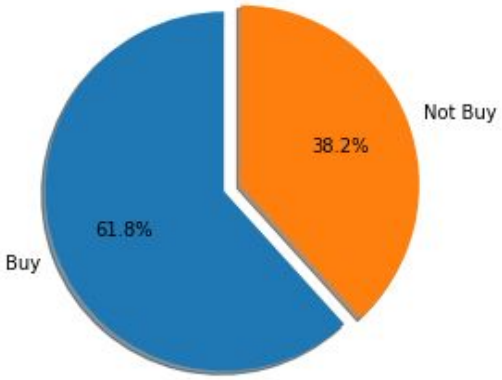
Items Bought when clicked maximum time in session



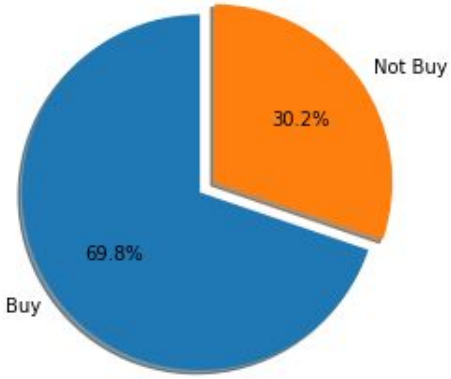
When Minimum time spent on an item in a session

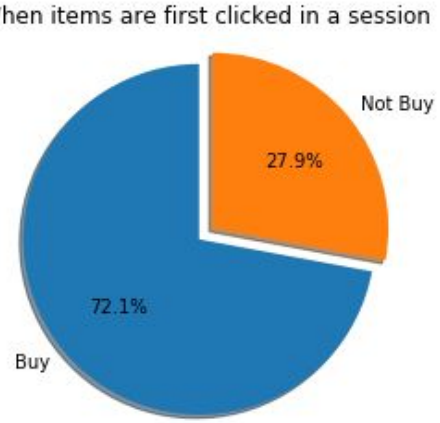


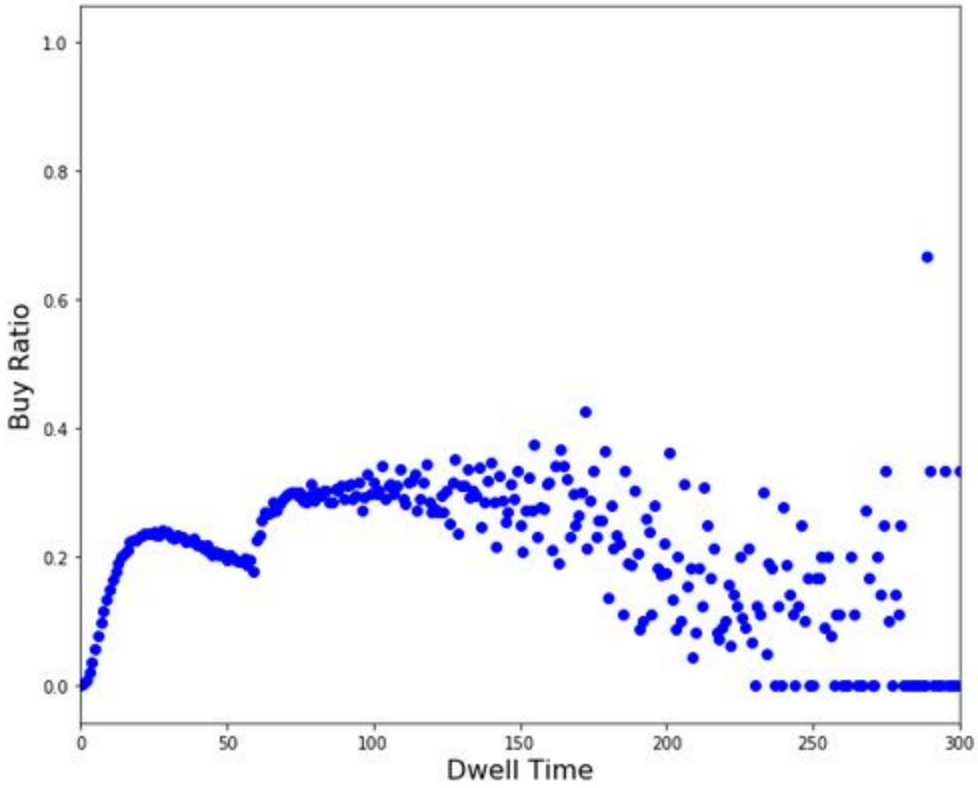
When Maximum time spent on an item in a session

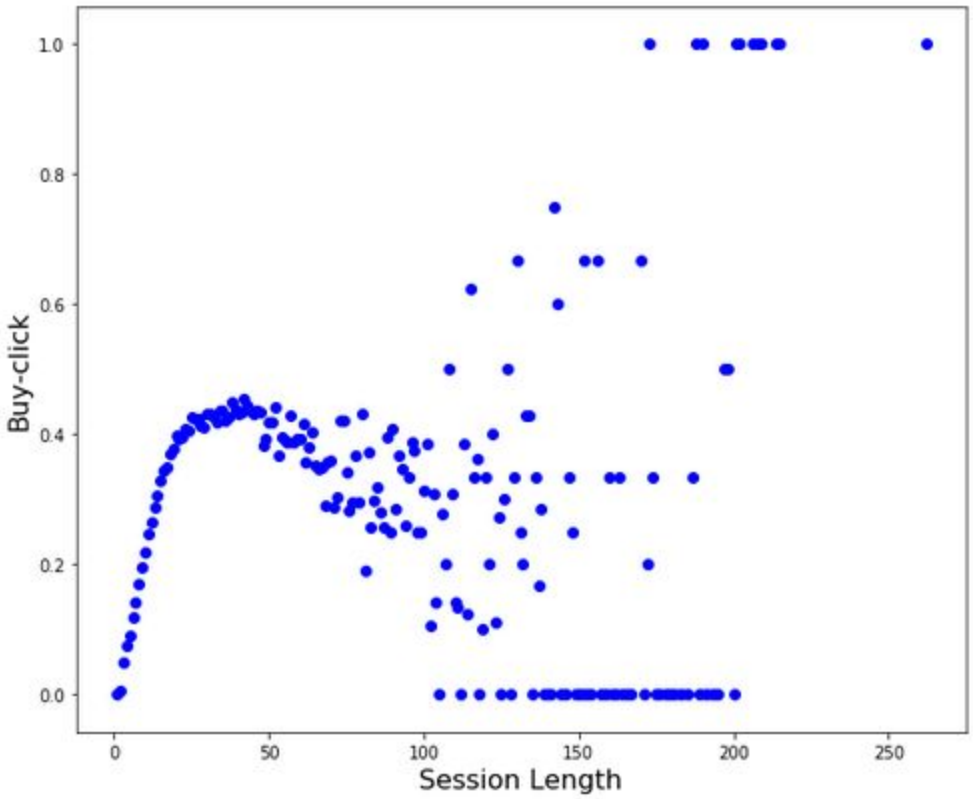


When items are last clicked in a session




When items are first clicked in a session
Not Buy
27.9%
72.1%
Buy


Buy Ratio
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250
300
Dwell Time



Exploratory analysis reveals that some dataset characteristics such as distributions of clicks on frequently purchased items, session durations, or number of item clicks demonstrate slightly distinguishing patterns for sessions with and without purchases.

This analysis leads us to engineer a mixture of categorical and numeric features for every item clicked in a session as illustrated in Table .

Feature	Type	Description
session	Integer	Session ID
item	Integer	Item ID
total_time	Integer	Total time spent on a session
avg.time_clicks	Real	Average time between the clicks
max_time	Integer	Maximum time between the clicks
n_clicks	Integer	Number of clicks in this session
avg_pop_score	Real	Average popularity Score of all items in this session
no_of_categories	Integer	Number of unique categories in this session
item_time	Integer	time spent on this item in this session
dow_first	Integer	Day of week of the first click on this item in this session
dow_last	Integer	Day of week of the last click on this item in this session
duration_f_l	Integer	Duration between the first click and the last click on this item in this session
f_click	Bool	Is this item the first click in this session
l_click	Bool	Is this item the last click in this session
item_pop	Real	popularity score of this item
purchased	Bool	whether this item is Purchased in this session

METHOD USED:

The fundamental approach we take to solve the two Challenge tasks relies on feature engineering, classification and boosting. For the two tasks, we first generate features to describe sessions and clicks. Then we apply a set of models to predict the accuracy .

Here are some of the models which we have used:

Gradient Boosting Tree (GBT) classifier:

Gradient boosting is typically used with decision trees (especially CART trees) of a fixed size as base learners. A gradient descent procedure is used to minimize the loss when adding trees. Gradient Boosting Trees build trees one at a time, where each new tree helps to correct errors made by previously trained tree.

Boosting helps us to focus on difficult examples that gives a nice strategy to deal with unbalanced datasets by strengthening the impact of the positive class. Since our data set is to imbalanced it helps us to balanced the not buying classes with the buying class. It helps us to learning to rank which means the construction of ranking models for highest precision without giving too many genuine examples to the model.

Random Forest Classifier:

Random forest classifier is an ensemble of decision trees which offers various sources of randomness to break correlations among predictor variables and also the trees in the ensemble. Typically, each tree works on a bagged version of the input dataset. Furthermore, each node split in each tree is decided by considering a random subset of features rather than the whole feature set. The expectation is that the variance of the final classifier is lower than individual classifiers while its bias is compensated. Random forest classifier also has additional benefits in our case: First, it can handle both categorical and real features. Furthermore, feature selection is inherent. Second, the training and testing of each tree in the ensemble are embarrassingly parallel which facilitates working with large datasets. Third, class weights can be incorporated to algorithm logic for cost-sensitive learning. Fourth, it can return class posterior probabilities which can be used for post-processing the results.

Naive Bayes Classifier:

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The crux of the classifier is based on the Bayes theorem.

In our data Naive Bayes Classifier is not working well because in our originated data frame there are too many values where the Not buying status is

very large than the buying status. Which helps Naive bayes to predict less. So for us Naive Bayes Classifier is not efficient.

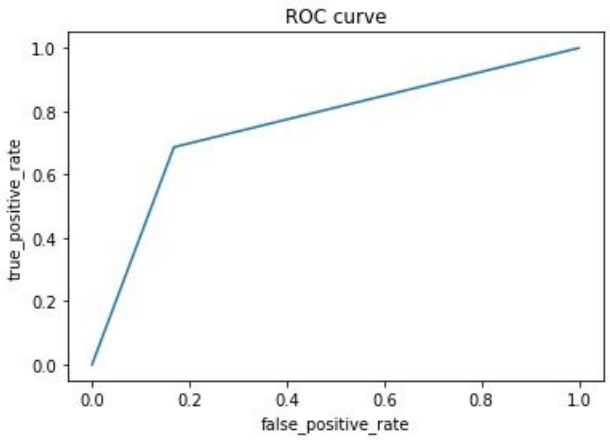
KNN:

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition . KNN is performing better than the

WORK PERFORMED:

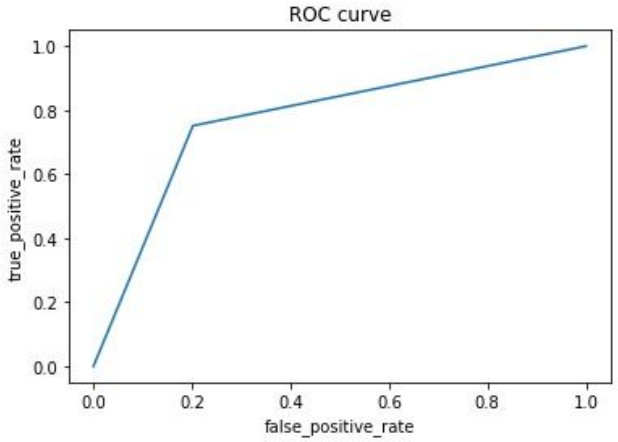
RESULTS:

ROC Curve and Accuracy of each of each of the model we have used:



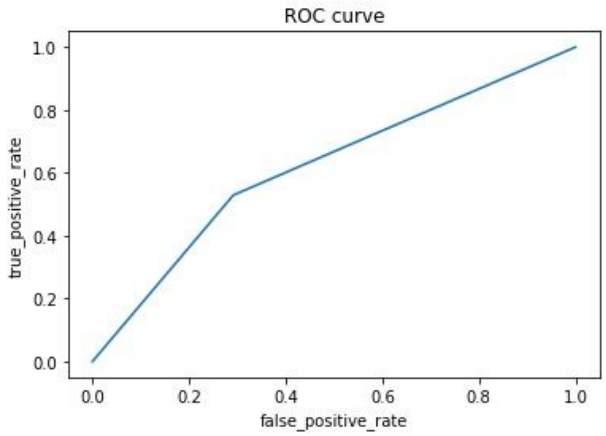

```
roc_auc 0.7593105789768534
accuracy_score 0.7660878506234106
```

RANDOM FOREST



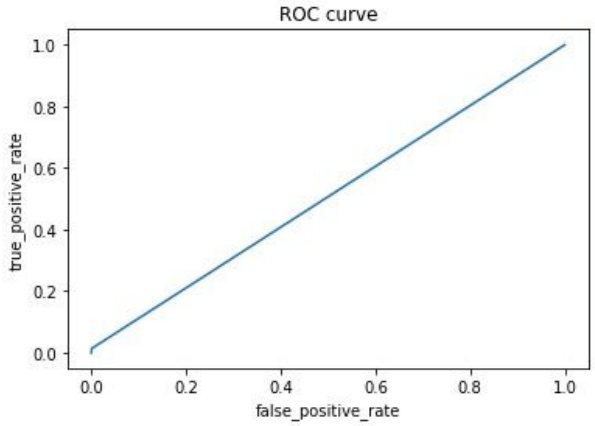

```
roc_auc 0.774755810379415
accuracy_score 0.7768884485445114
```

GRADIENT BOOSTING




```
roc_auc 0.6187691013149763
accuracy_score 0.6273131280547691
```

KNN




```
roc_auc 0.5064393293361613
accuracy_score 0.5523540983228832
```

NAIVE BAYES

TASK ASSIGNMENT:

Task Name	Contributions			
	Abhinav	Abhijeet	Shubham	Nishant
Data Exploration	20	20	30	30
Data Preparation & Feature Extraction	10	70	10	10
Data Modelling ->				
Random Forest	70	10	10	10

Gradient Boosting	**20**	**10**	**20**	**50**
KNN	**40**	**40**	**10**	**10**
Naive Bayes	**10**	**10**	**60**	**20**

CHALLENGES:

- Data not directly usable: needed to extract new features in order to get noticeable accuracy.
- Huge Datasets made Analysis extremely slow
- Feature Extraction is not limited, Many more features could be extracted for further improving the performance.
- Class Imbalance Problem

GITHUB LINK: https://github.com/abnvanand/smai-proj