Two-Stage Approach to Item Recommendation from User Sessions

Maksims Volkovs
Milq Inc.
151 Bloor Street West
Toronto, ON M5S 1S4
maksims.volkovs@gmail.com

ABSTRACT

We present our solution to the 2015 RecSys Challenge [1]. This challenge was based on a large scale dataset of over 9.2 million user-item click sessions from an online e-commerce retailer. The goal was to use this data to predict which items (if any) were bought in the 2.3 million test sessions. Our solution to this problem was two-staged, we first predicted if a given session contained a buy event and then predicted which items were bought. Both stages were fully automated and used classifiers trained on large sets of extracted features. The prediction rules were further optimized to the target objective using a greedy procedure developed specifically for this problem. Our best submission, which was a blend of several different models, achieved a score of 60,265 and placed 4'th out of 567 teams. All approaches presented in this work are general and can be applied to any problem of this type.

Categories and Subject Descriptors

H.4.m [**Information Systems Applications**]: Miscellaneous

Keywords

E-Commerce; Classification; Neural Networks; Gradient Boosting

1. CHALLENGE OVERVIEW

In this challenge, a large dataset of user sessions was provided from an online retailer. Each session contained user-item click history including time stamp, item id and item category for all clicked items. In addition to session information, item buy events were also released for a subset of sessions. Each buy event contained a corresponding session id, bought item id, time stamp, quantity bought and price. Data was further anonymized by removing all user information. The goal of the challenge was to predict buy events for the test sessions. In addition to predicting which test sessions contained buy events, a second requirement was to predict which items were bought.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
RecSys'15, September 16 - 20, 2015, Vienna, Austria.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3665-9/15/09-.$15.00
DOI: http://dx.doi.org/10.1145/2813448.2813512.

Table 1: Dataset statistics for the 2015 RecSys Challenge. Number of sessions containing at least one buy event are shown in brackets. train_split and valid_split were generated by applying 75%/25% split to the original training set (train). These sets were used for training and validation throughout the challenge.

dataset	sessions	item clicks	buy events
train	9,249,729	33,003,944	1,150,753 (509,696)
test	2,312,432	8,251,791	
train_split	6,937,297	24,753,225	862,539 (382,256)
valid_split	2,312,432	8,250,719	288,214 (127,440)

Full dataset statistics are shown in Table 1. From the table we see that this dataset is large and extremely imbalanced. Only 5.5% of training sessions contain buy events, and only 3.5% of clicked items were bought. Dealing with imbalanced classes and optimizing for the custom evaluation metric were the two main challenges that we encountered throughout this competition. In the following sections we describe our approach and show empirical evaluation of proposed methods.

2. FRAMEWORK

We begin with notation and evaluation framework. In the data we have a set of N sessions $\mathbf{S} = \{s_1, ..., s_N\}$, and each session s_n contains M_n clicked items $\mathbf{D}_n = \{d_{n1}, ..., d_{nM_n}\}$. Note that $\mathbf{D}_n$ is a set and only contains unique items. We also have access to the ground truth buy event data $\mathbf{Y}_n = \{y_{n1}, ..., y_{nM_n}\}$ where $y_{nm} = 1$ if item d_{nm} was bought in s_n and $y_{nm} = 0$ otherwise. We use $y_n = 1$ to indicate that session s_n has at least one buy event and $y_n = 0$ otherwise. Matrix/vector notation is used throughout this paper: for any matrix a, $a[i,j]$ is the element on i'th row and j'th column, and $a[i,:]$ is the i'th row of a. Similarly for any vector b, $b[i]$ is the i'th element of b.

The goal of the challenge was to develop a model over items $f : d \rightarrow \{0, 1\}$, such that f has maximal "agreement" with the ground truth $\mathbf{Y}$. The agreement between f and $\mathbf{Y}$ is evaluated with a custom metric introduced specifically for this challenge:

$$\mathcal{O} = \sum_{n=1}^{N} \begin{cases} \gamma + \frac{\sum_{m=1}^{M_n} y_{nm} f(d_{nm})}{\sum_{m=1}^{M_n} y_{nm} + f(d_{nm}) - y_{nm} f(d_{nm})} & \text{if } y_n = 1 \\ -\gamma & \text{otherwise} \end{cases} \quad (1)$$

where γ is a constant set to the fraction of test sessions that have at least one buy event. Since buy events for test ses-

sions were not released, this constant was unknown throughout the competition and had to be estimated. We estimate it by randomly sampling the same number of training sessions as in the test set, obtaining the following estimate: $\gamma \approx 0.05511$.

Note that both $f(d_{nm})$ and y_{nm} are binary, so the numerator in Equation 1 corresponds to the intersection of predicted and bought items. Similarly, the denominator corresponds to the union of predicted and bought items, this objective thus computes the intersection over union Jaccard score for every session s_n. Furthermore, if the predicted session s_n has at least one buy event, the score for s_n is always at least γ even when all predicted items are wrong.

3. OUR APPROACH

We started with an observation that the target objective $\mathcal{O}$ is not smooth and cannot be optimized directly with gradient-based methods. 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 sessions and items that maximize $\mathcal{O}$. We chose to use a probabilistic approach and generate predictions in the following way:

$$f(d_{nm}) = \begin{cases} 1 & \text{if } P(y_n = 1) > \epsilon \text{ and } P(y_{nm} = 1) > \beta \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Here, ϵ and β are thresholds applied to probability predictions of session model $P(y_n = 1)$ and item model $P(y_{nm} = 1)$ respectively. In practice, this procedure can be short circuited and terminated early if session probability does not pass the threshold, leading to considerable savings in inference time. In the following sections we describe both model architectures and our greedy inference procedure used to select the thresholds.

3.1 Session Model

Very little information was provided beyond item and session ids, so our initial approach was to use collaborative filtering. We represented each session as a hypothetical user and experimented with both neighbor and factorization approaches to predict which items were bought. The accuracy of these methods was not adequate and as we moved on to supervised feature-based classification. In this framework the probability that s_n contains at least one buy event is modeled with a sigmoid:

$$P(y_n = 1) = \frac{1}{1 + \exp(-g_s(s_n, \theta_s))} \quad (3)$$

where $g_s : s_n \to \mathcal{R}$ is a parametrized by θ_s session model applied to features extracted for s_n.

At this stage most effort was spent on feature engineering and after many rounds of cross validation we selected a total of 37 session features outlined in Appendix A [1]. Throughout this process we consistently found that "global" features that included information such as the number of times items in s_n were bought/clicked in other sessions were most predictive. Here we give an example of one such feature:

$$\sum_{m=1}^{M_n} \frac{\text{buy}(\cdot, d_{nm})}{\text{click}(\cdot, d_{nm}) + \beta} \times \text{click}(s_n, d_{nm})$$

where $\text{buy}(\cdot, d_{nm})$ and $\text{click}(\cdot, d_{nm})$ are the number of training sessions where d_{nm} was bought and clicked respectively; $\text{click}(s_n, d_{nm})$ is the number of times d_{nm} is clicked in s_n and $\beta = 10$ is a smoothing constant. The ratio $\text{buy}(\cdot, d_{nm})/(\text{click}(\cdot, d_{nm}) + \beta)$ can be interpreted as the probability estimate that d_{nm} is going to be bought in any session where it is clicked. The additional weighting by $\text{click}(s_n, d_{nm})$ (another option is to weight by time spent) emphasizes items that user found particularly interesting in s_n. This feature thus combines the global likelihood of buying d_{nm} with specific user interest for d_{nm} in s_n. We found that the correlation between this features and $\mathbf{Y}$ was 0.28 – the highest of any session feature that we experimented with. It is worth mentioning here that features of this type resemble TFIDF statistics commonly used in text analysis, classification and clustering. Analogous features are also frequently used in web search personalization from user sessions [3, 2], which is a very similar problem to the one presented in this challenge.

In addition to manually derived features, we found that it was very useful to include an indicator feature with all the items that were clicked in a given session. Since there were only 54,287 unique items in the entire dataset, we were able to include all items (excluding those not present in the training data) in this indicator feature. Formally, for every session s_n, we created a $1 \times 54,287$ indicator vector I_n where for $1 \leq m \leq 54,287$, $I_n[m] = 1$ if $d_{nm} \in \mathbf{D}_n$ and $I_n[m] = 0$ otherwise. Incorporating indicator features allows the model to have item-specific parameters and learn correlations between groups of clicked items and buy events. All of our sessions models were trained on concatenated vectors of features from Appendix A and I_n. Note that this approach can be applied even when the number of items is large by using the hashing trick [4] or selecting only the most popular items. We experimented with the latter method and found that comparable accuracy can be achieved by using only the top 5,000 most popular items.

3.2 Item Model

After obtaining encouraging results with feature-based session models we decided to take an analogous approach for item prediction. The probability that a given item d_{nm} was bought in s_n was similarly modeled by a sigmoid:

$$P(y_{nm} = 1) = \frac{1}{1 + \exp(-g_d(d_{nm}, \theta_d))} \quad (4)$$

where $g_d : d_{nm} \to \mathcal{R}$ is a parametrized by θ_d item model applied to features extracted for d_{nm}.

In this stage the most effort was also spent on feature engineering. After several rounds of feature selection we ended up with a set of 20 features summarized in Appendix B. Through these experiments we found that features which contained global buy and click information for a given item were also the most predictive. In addition to item-specific features, we also found it useful to include the corresponding session information as input to the item model. For a given item d_{nm} we first tried to incorporate the output of the session model $P(y_n = 1)$ by appending it to d_{nm}'s features. This improved the accuracy but often led to highly

[1] Full appendix with all the features can be found at `www.cs.toronto.edu/~mvolkovs/recsys2015.pdf`.

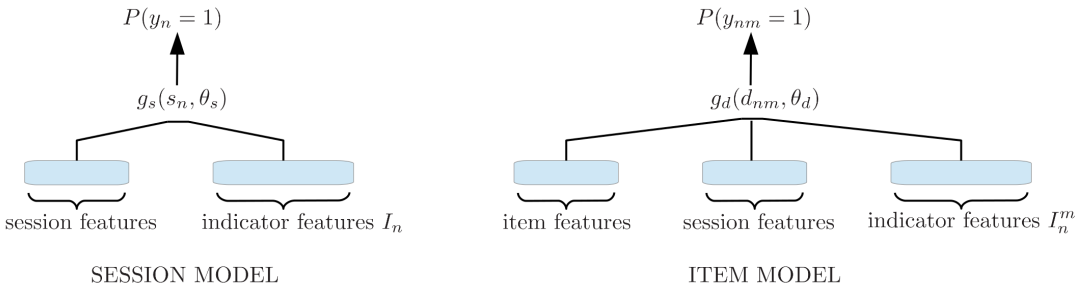

Figure 1: Input/output diagrams for session and item models. Both models use the same session features; item model also incorporates item-specific features and slightly modified indicator vector I_n^m (see Equation 5).

over-fitted solutions. Our second approach was to pass raw session features (including indicator) as additional input to the item model. This turned out to work very well and together with item features, was the basis of our item model throughout the competition.

In the final stages of the competition we also experimented with various ways to incorporate current item info into the model. I_n doesn't distinguish between items that were clicked in s_n. While this is appropriate for the session model, for the item model we wanted to further indicated which of clicked items the model is currently generating the prediction for. To achieve this, we introduced the item-specific indicator I_n^m for each item d_{nm}:

$$I_n^m[i] = \begin{cases} -1 & \text{if } i = m \\ 1 & \text{if } i \neq m \text{ and } d_{ni} \in \mathbf{D}_n \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Note that I_n^m is almost identical to I_n but contains -1 in the target position m, thus allowing the model to identify the item. We consistently found that using I_n^m in place of I_n produced better predictions, although the gains were not very significant. Figure 1 summarizes the input/output architectures for both session and item models with all the feature components that were included in the final versions of each model.

3.3 Inference

We mentioned above that by thresholding session and item probabilities we can generate more confident predictions. However, it is unclear how to choose these thresholds such that the resulting predictions maximize $\mathcal{O}$. In this section we outline a greedy procedure that we developed to solve this problem.

We first note that the contribution from each session to the overall score is highly dependent on the number of clicked items in that session. The more items a given session has the harder it is to maximize the Jaccard score in Equation 1. This suggests that session probability thresholds should depend on session size. Second, the accuracy of the item model can vary significantly depending on the position of the item. If items for each session are sorted by the probability from the item model, then the top ranked predictions would tend to be more accurate than those ranked 2'nd, 3'rd etc. This also suggests that item probability thresholds should depend on the item position in the sorted by probability ranking.

Combining these observations, we developed an iterative greedy procedure where we select a separate session probability threshold and a set of item probability thresholds for each session size. We first sort item probabilities for each session in descending order and use $r_{nm} \in \{1, 2, ..., M_n\}$ to denote the rank of item d_{nm} in this sorted order. We use

Algorithm 1 Greedy Inference

Input: session size k
candidate session thresholds $E = \{a_1, a_2, ...\}$
candidate item thresholds $B = \{b_1, b_2, ...\}$
for $i = 1$ **to** $|E|$ **do**
  $\alpha[k] = a_i$
  $\beta[k, :] \equiv 0$
  for $t = 1$ **to** k **do**
    select threshold from B for $\beta[k, t]$ that maximizes $\mathcal{O}_t^k$
  end for
  store $\alpha[k]$ and $\beta[k, :]$ if higher $\mathcal{O}_k^k$ is reached
end for
Output: $\alpha[k]$ and $\beta[k, :]$ with highest $\mathcal{O}_k^k$

$d_{r_{nm}}$ to denote the item in position r_{nm} with the corresponding target $y_{r_{nm}}$. We then modify the prediction rule in Equation 2 to incorporate session and ranking dependent thresholds:

$$f(d_{nm}) = \begin{cases} 1 & \text{if } P(y_n = 1) > \alpha[M_n] \text{ and } P(y_{nm} = 1) > \beta[M_n, r_{nm}] \\ 0 & \text{otherwise} \end{cases}$$

Here, α is now a vector of thresholds and $\alpha[M_n]$ is a threshold for session with M_n items. Similarly, β is now a matrix of thresholds and $\beta[M_n, r_{nm}]$ is a threshold for item ranked in position r_{nm}. Note that the same α and β are used for *all* sessions with the same number of items.

The thresholds are selected using a greedy approach where at each step we maximize the *partial objective*:

$$\mathcal{O}_t^k = \sum_{s_n : M_n = k} \begin{cases} \gamma + \frac{\sum_{i=1}^{t} y_{r_{ni}} f(d_{r_{ni}})}{\sum_{i=1}^{k} y_{r_{ni}} + \sum_{j=1}^{t} f(d_{r_{nj}}) - y_{r_{nj}} f(d_{r_{nj}})} & y_n = 1 \\ -\gamma & \text{ow.} \end{cases}$$

$\mathcal{O}_t^k$ only considers sessions with k items, and for each such session it only evaluates the top-t items predicted by the item model. Decomposing $\mathcal{O}$ into O_t^k's allows us to conduct sequential optimization where for all sessions of size k we: 1) fix session threshold $\alpha[k]$ 2) sequentially find item thresholds $\beta[k, 1]$, $\beta[k, 2]$,... that maximize $\mathcal{O}_1^k$, $\mathcal{O}_2^k$,... 3) check if higher O_k^k is reached. Steps 1-3 are repeated for all candidate thresholds, and settings that maximize O_k^k are returned. Algorithm 1 summarizes this procedure. We run Algorithm 1 in parallel for all session sizes to produce the final α β configurations and use them to generate test set predictions. It is important to note here that all threshold selection is done on the validation set to avoid biased solutions from potentially over-fitted models.

Empirically, we found that selecting separate thresholds for sessions with more than 15 items didn't generalize well

Table 2: Validation (valid_split) and leaderboard results. NN\I excludes indicator features from both session and item models. Pruned NN and GBM models were retrained after removing all sessions that were not selected by the session thresholds α.

model	valid_split	leaderboard
NN\I	43,994	52,822
NN	49,237	58,594
GBM	49,688	58,820
Pruned NN	50,493	59,826
4. Pruned NN+GBM	**50,793**	**60,265**
top-5 results from other teams		
1. Peter		63,102
2. Cloud Card		62,656
3. Random Walker		61,075
5. Budapest		59,845
6. Tøyvind thørrud		55,078

since there are too few such sessions. To deal with this we cap k at 15 and for $k = 15$ consider all sessions with 15 or more items. Similarly, we found that predicting more than 7 items per session did not produce significant gains. So we also cap the maximum number of items predicted for any session to 7. Both of these constraints were incorporated into Algorithm 1 to generate our final solutions.

4. EXPERIMENTS

In this section we describe our experimental set-up and results. For all experiments we used a single 75%/25% training/validation split (see Table 1) to train and tune all models. We primarily concentrated on neural net (NN) and gradient boosting (GBM) classifiers for both session and item models.

For NN classifiers, we used cross entropy (logistic regression) as the target objective and validation AUC for early stopping. AUC is well suited for this task since it is not sensitive to large class imbalances. We optimized all NN models with minibatch stochastic gradient descent. Minibatch size was set to 20K for both session and item models, and we always ensured that each minibatch contained 10K positive and 10K negative examples. Having equal class proportions allowed for more stable optimization and we found it to work at least as well as any other class proportion that we tried. We experimented with both linear and non-linear (1, 2 and 3 hidden layers) NNs and found that 1-hidden layer models gave the best performance. For non-linear models, given that the feature vectors have more than 50K dimensions, even small models with 100 hidden units in the first hidden layer have more than 5M parameters. The number of parameters thus far exceeded the number of positive examples, and very strict regularization was required to avoid over-fitting. We experimented with L^1, L^2 and dropout to prevent over-fitting and found L^2 to work the best. Dropout also gave good performance but was extremely slow to converge.

For GBM classifiers we used the excellent XGBoost [2] library. Similarly to NNs, we concentrated on logistic regression GBMs with tree boosters and used validation AUC for early stopping. GBM is even more prone to over-fitting so we spent considerable effort tuning the parameters of XGBoost to prevent over-fitting. We found the following four parameters to be particularly useful: tree depth (`max_depth`), learning rate (`eta`), min weight to split a tree node (`min_child_weight`) and data sub-sampling (`subsample`). For both session and item models we set `max_depth` in [10, 20], `eta` in [0.1, 0.01], `min_child_weight` in [50, 200] and `subsample` in [0.7, 0.9]. We generally found that smaller learning rates produced better models at the expense of longer training times. On a server with 16 Intel Xeon 2.90GHz CPUs and 64GB of RAM, NN generally took several hours to train for both session and item models whereas GBMs (using 4 threads) took about 1.5 days.

[2] `https://github.com/dmlc/xgboost`

4.1 Results

Table 2 shows validation and leaderboard results for some of the better models that we experimented with. Several patterns can be seen from this table, first, we consistently found that major improvements on the validation set always translated to the test set. This suggests that one validation split is enough for all model tuning, allowing to avoid expensive n-fold validation. Second, we found indicator features to be very useful with score improvements of over 5K for models that included these features (NN\I vs NN). Third, given that very few sessions have buy events and that models have limited capacity, we introduced an additional pruning step removing session that didn't pass the cut-off. After training both session and item models and selecting appropriate α and β thresholds, we used α to remove all sessions from training, validation and test sets that didn't pass this threshold. This removed over 70% of sessions from each of the 3 sets and we then re-trained the models on the significantly reduced training data. From Table 2 we see that "Pruned NN" which was re-trained on the pruned training data produced gains of up to 1K while taking significantly less time to train. This suggests a general approach for imbalanced classification where irrelevant data is iteratively removed allowing the model to concentrate on the important training examples. We plan to explore this further in the future.

Our best submission with a score of 60,265 achieved 4'th place using a blend of several NN and GBM models trained on the pruned subset of the training data. We experimented with several blending techniques but found that simple probability averaging produced the best results. It was surprising that blending gave only marginal improvement. After further analysis we think that this can be attributed to the disconnect between the cross entropy/AUC objectives optimized during training and the target $\mathcal{O}$. We often saw cases where improvement in AUC did not translate to improvement in $\mathcal{O}$.

5. REFERENCES

[1] D. Ben-Shimon, A. Tsikinovsky, M. Friedman, B. Shapira, L. Rokach, and J. Hoerle. Recsys challenge 2015 and the yoochoose dataset. In *ACM RecSys*, 2015.

[2] P. Masurel, K. Lefèvre-Hasegawa, C. Bourguignat, and M. Scordia. Dataiku's solution to yandex's personalized web search challenge. In *WSDM*, 2014.

[3] J. Teevan, S. T. Dumais, and D. J. Liebling. To personalize or not to personalize: Modeling queries with variation in user intent. In *SIGIR*, 2008.

[4] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *ICML*, 2009.

APPENDIX

A. SESSION FEATURES

After many rounds of cross validation we used a total of 37 session features for each session s_n. To extract the features we use the following helper functions:

buy(s_n, d_{nm}): number of times d_{nm} was clicked in s_n.

click$(\cdot, d_{nm})$: number of training sessions where d_{nm} was clicked.

buy$(\cdot, d_{nm})$: number of training sessions where d_{nm} was bought.

sim_click(d_{ni}, d_{nj}): similarity between d_{ni} and d_{nj} using collaborative filtering item-based neighborhood similarity on session-item click matrix. Higher sim_click score indicates that d_{ni} and d_{nj} are frequently clicked together.

sim_buy(d_{ni}, d_{nj}): similarity between d_{ni} and d_{nj} using collaborative filtering item-based neighborhood similarity on session-item buy matrix. Higher sim_buy score indicates that d_{ni} and d_{nj} are frequently bought together.

$\mathbf{d_{nM_n}}$: last clicked item in s_n.

Session statistics

1. s_n duration in milliseconds
2. number of clicks in s_n
3. number of unique items clicked
4. max time spent on any item
5. cross entropy of "time spent" distribution across items

6-7. number of items with 2 clicks and ≥ 3 clicks

8-10. number of items with category 0, -1 and in $(0, 12]$

Global statistics The goal here was to aggregate buy and click information for items in s_n from all training sessions where these items appear. We then combine this "global" information with "local" click patterns in s_n to estimate the likelihood of s_n containing a buy event.

11. $\sum_{m=1}^{M_n} \text{click}(\cdot, d_{nm})$
12. number of unique items in s_n that were previously bought in other training sessions
13. $\sum_{m=1}^{M_n} \text{buy}(\cdot, d_{nm})$
14. $\sum_{m=1}^{M_n} \text{buy}(\cdot, d_{nm}) \times \text{click}(s_n, d_{nm})$
15. $\max_{d_{nm}} \frac{\text{buy}(\cdot, d_{nm})}{\text{click}(\cdot, d_{nm})+\beta}$
16. $\sum_{m=1}^{M_n} \frac{\text{buy}(\cdot, d_{nm})}{\text{click}(\cdot, d_{nm})+\beta}$
17. $\sum_{m=1}^{M_n} \frac{\text{buy}(\cdot, d_{nm})}{\text{click}(\cdot, d_{nm})+\beta} \times \text{click}(s_n, d_{nm})$
18. $\text{click}(\cdot, d_{nM_n})$
19. $\frac{\text{buy}(\cdot, d_{nM_n})}{\text{click}(\cdot, d_{nM_n})+\beta} \times \text{click}(s_n, d_{nM_n})$

Collaborative filtering item similarity
The motivation behind these features is that users who are looking to buy are more likely to do targeted browsing. Targeted browsing often involves items that are frequently bought together such as items from the same category/type. Sessions where many items are "similar" and are often clicked/bought together by other users are thus more likely to contain a buy.

20. $\sum_{i>j} \text{sim\_click}(d_{ni}, d_{nj})$
21. $\sum_{i>j} \text{sim\_click}(d_{ni}, d_{nj}) \times (\text{click}(s_n, d_{ni}) + \text{click}(s_n, d_{nj}))$
22. $\max_{i>j} \text{sim\_click}(d_{ni}, d_{nj})$
23. $\max_{i>j} \text{sim\_click}(d_{ni}, d_{nj}) \times (\text{click}(s_n, d_{ni}) + \text{click}(s_n, d_{nj}))$
24. $\text{std}_{i>j} \text{sim\_click}(d_{ni}, d_{nj})$

Time features
From initial data inspection we found evidence of seasonality where number clicks/buys changed depending on day of the week and hour of the day [3]. Time features were aimed at capturing these seasonality effects.

25-31. day of the week indicator

32-37. hour of day indicator grouped into six four-hour intervals 0-4, 4-8, 8-12 etc.

B. ITEM FEATURES

Similarly to session features, after several rounds of cross-validation we selected a total of 20 item features:

Session statistics

1. $\text{click}(s_n, d_{nm})$
2. time spent on d_{nm} in milliseconds
3. 1 if d_{nm} was clicked last in s_n and 0 otherwise

4-6. item category indicator for categories: -1, 0, and in (0, 12]

Global statistics

7. $\text{click}(\cdot, d_{nm})$
8. $\text{buy}(\cdot, d_{nm})$
9. $\text{buy}(\cdot, d_{nm}) \times \text{click}(s_n, d_{nm})$
10. $\frac{\text{buy}(\cdot, d_{nm})}{\text{click}(\cdot, d_{nm})+\beta}$
11. $\frac{\text{buy}(\cdot, d_{nm})}{\text{click}(\cdot, d_{nm})+\beta} \times \text{click}(s_n, d_{nm})$

Collaborative filtering item similarity
A given item is more likely to be bought if corresponding session contains other items that are similar (frequently co-clicked and co-bought together). Similar items indicate that the user was doing targeted browsing and was looking for a particular category/type of items which in turn increases the likelihood of a buy.

12. $\sum_i \text{sim\_click}(d_{nm}, d_{ni})$
13. $\max_i \text{sim\_click}(d_{nm}, d_{ni})$
14. $\text{std}_i \text{sim\_click}(d_{nm}, d_{ni})$
15. $\sum_i \text{sim\_buy}(d_{nm}, d_{ni})$
16. $\max_i \text{sim\_buy}(d_{nm}, d_{ni})$
17. $\text{std}_i \text{sim\_buy}(d_{nm}, d_{ni})$
18. $\sum_i \frac{\text{sim\_buy}(d_{nm}, d_{ni})}{\text{sim\_click}(d_{nm}, d_{ni})} \times (\text{click}(s_n, d_{nm}) + \text{click}(s_n, d_{ni}))$
19. $\max_i \frac{\text{sim\_buy}(d_{nm}, d_{ni})}{\text{sim\_click}(d_{nm}, d_{ni})} \times (\text{click}(s_n, d_{nm}) + \text{click}(s_n, d_{ni}))$
20. $\text{std}_i \frac{\text{sim\_buy}(d_{nm}, d_{ni})}{\text{sim\_click}(d_{nm}, d_{ni})} \times (\text{click}(s_n, d_{nm}) + \text{click}(s_n, d_{ni}))$

[3]Similar patterns were found by Humberto Corona and summarized in this blog post: `http://aloneindecember.com/words/recsys-challenge-part-ii/`