CS771: Assignment 2 (Group 31)

Aniket Sanghi
Roll no: 170110
sanghi@iitk.ac.in

Paramveer Raol
Roll no:170459
paramvir@iitk.ac.in

Sarthak Singhal
Roll no:170635
ssinghal@iitk.ac.in

Mihir Jewalikar
Roll no:170387
mihirsj@iitk.ac.in

Abhishek Bhatia
Roll no:170022
avishek@iitk.ac.in

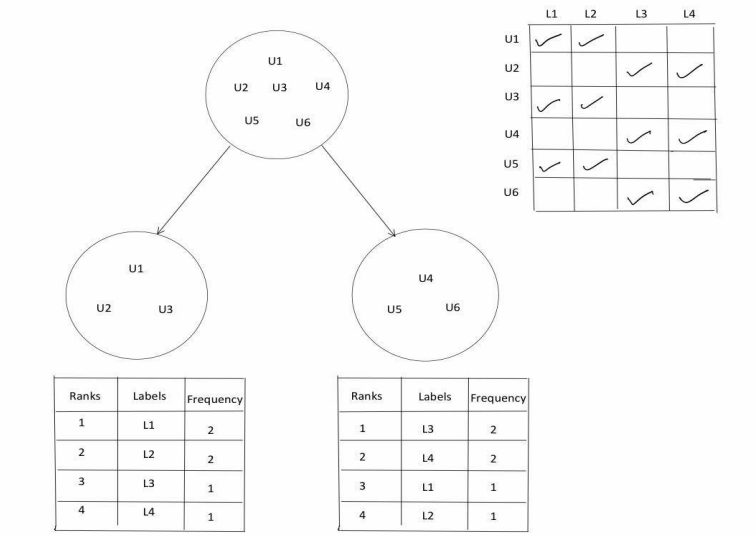
1 Part 1: FastXML[1]

Abstract : FastXML[1] uses decision trees for the extreme classification. At each internal node it uses a linear binary classifier to classify the test point to either left or the right child. Linear classifier is learnt by using a loss function which uses the ranking of the labels as the criteria. For predicting the labels' ranking for the given feature vector, it first traverses each of the trees in the model and the final ranking is obtained by averaging out the rankings obtained from the leafs reached in each tree.

Building a Tree[3]

Node Partitioning
At the time of training each node corresponds to a set of users. Each node also represents the ranking of the labels where the ranking is defined by the number of users in the node that like that label. The Node Partitioning involves the following steps -

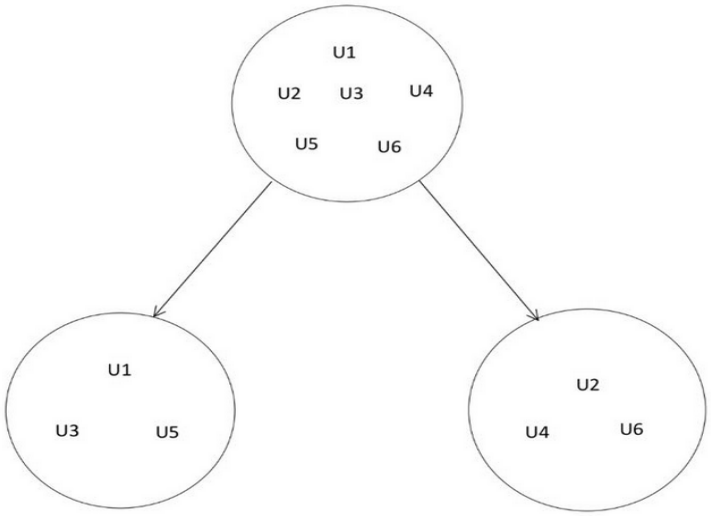
The first step is to randomly split users into 2 mutually exclusive set. Afterwards rank the labels in each of the child nodes based on the number of users in the child node that like that particular label. In order to find the optimal partitioning the following Alternating Optimisation is



used -

Preprint. Under review.

1. Fix the ranking list of the two child nodes.
For each user compare the ranking of the items that the user prefers and switch accordingly (ie. the user remain in same child or it goes in the other child).

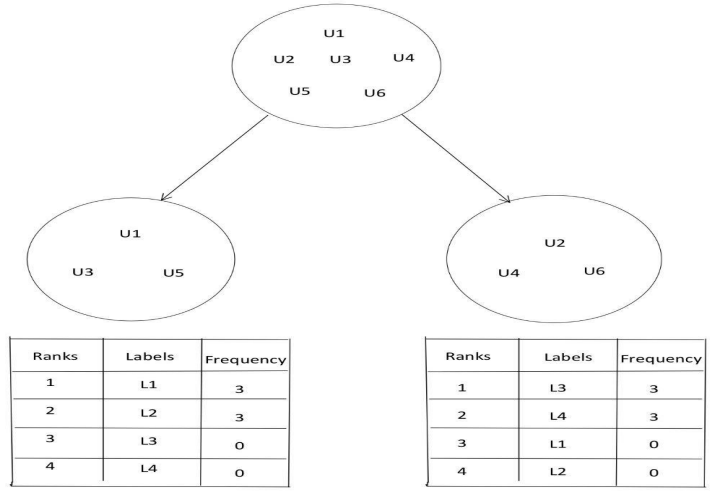


In order to compare ranking for user nDGC is incorporated into the loss function:[3][1]

$$\text{nDGC} \propto like(i, r_1) + \sum_{l=2}^{L} \frac{like(i,r_l)}{log(l+1)}$$

$$like(i, r_l) = \begin{cases} 1 \text{ if user i like the item with rank } r_l \\ 0 \text{ otherwise} \end{cases}$$

2. Fix the users in two child nodes.
Now update the ranking list of the respective child nodes.



Ranks	Labels	Frequency
1	L1	3
2	L2	3
3	L3	0
4	L4	0

Ranks	Labels	Frequency
1	L3	3
2	L4	3
3	L1	0
4	L2	0

Termination : If the switching of users ceases,then convergence has occurred. This means more processing is not required and termination has been achieved.

The loss function employed for the process is:

$$\underset{\mathbf{w},\boldsymbol{\delta},\boldsymbol{r}^{\pm}}{\arg\min}\ \{\ ||\mathbf{w}||_2^2\ +\ \sum_{i=1}^{n} C_\delta(\delta_i) log(1+e^{-\delta_i<\mathbf{w},\mathbf{x}_i>}) - C_r \sum_{i=1}^{n} nDGC(r^{\delta_i})^t N_{y_i} y_i\ \}$$

Note: In this case n is the number of users.
In this loss function :

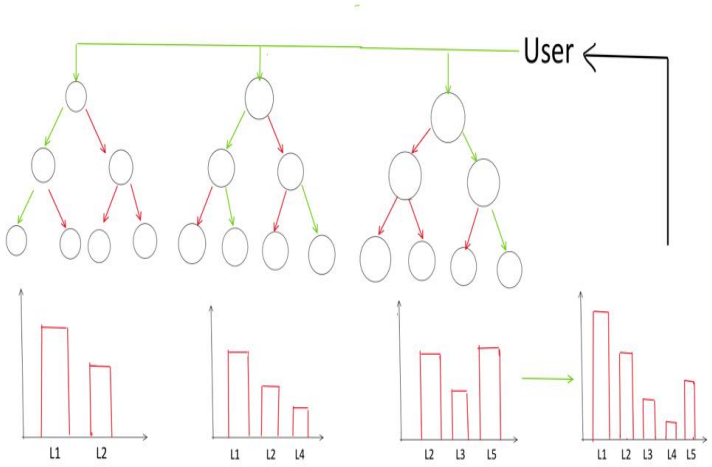
$\boldsymbol{\delta}$ for a given user's index. It is +1 if the user goes to the right child and -1 if the user goes to the left child.
$\boldsymbol{r}^{\pm}$ is the rank vector of items. $\mathbf{r}^{+}$ and $\mathbf{r}^{-}$ are the rank list of items for the right and left child respectively.

While using Alt-Opt, the first step is to improve the third term of the loss functions, then in the second step, the second and the third term together optimize and finally in the last part where linear binary classification model has to be found the second and the first term's combined value is minimized.

Termination of Splitting of node
One of the hyper-parameter in the tree building is the number of users in the leaf so as soon as number of users corresponding to one of the child nodes becomes less than or equal to that number no further splitting of the child node takes place.

Predicting items
An ensemble of trees is learnt in this FastXML$^{(1)}$, the number of trees to be learnt is the hyper-parameter. When a test feature vector is passed to the model then the traversal is done in each and every tree. A leaf will be selected from every tree and then the rank-vector of items would be decided in the following manner.

$$\mathbf{r}(\mathbf{x}) = \text{rank}_k(\tfrac{1}{T}\sum_{t=1}^{T} \mathbf{P}_t^{leaf}(\mathbf{x}))$$



2 Part 2

Advantages

1) Prediction time is quite fast as it finds the ranking by traversing a tree which takes logarithmic time.

2) Since the construction of each tree can be done independently and hence many concurrent threads can be used to achieve a faster training time. The authors of fastXML[(1)] trained it on single core, training of more than a million labels was done in 8 hrs whereas on multiple cores it took merely an hour.

3) It achieves more precision as it uses a ranking based loss function and not the conventional loss functions.

Disadvantages

1) FastXML[(1)], unlike PfastreXML does not give any weight to labels. Thus a popularity bias is created and rare labels which are way down in ranking will never be selected. i.e. rare items will never be recommended.

2) Large amount of trees need to be stored so even adding a single new feature would prove expensive.

3 Part 3

After running FastXML[(1)] on the data provided using the default hyper Parameters for training and testing, the following values were obtained by predict.py:

prec@1 = 0.873

prec@3 = 0.774

prec@5 = 0.678

mprec@1 = 0.001599

mprec@3 = 0.01219

mprec@5 = 0.03142

4 Part 4

a)

The main issue we faced while implementing FastXML[(1)] was that values of mprec were too low.The reason behind this is that FastXML assigns equal weights to both rare and popular items. Hence it creates a popularity bias and will not recommend rare items. We observed that PfastreXML[(2)] has much better performance in terms of mprec. In PfastreXML, weights are assigned to each item. The weights are inversely proportional to the popularity of the item. Thus it recommends rare items also. Formula used for assigning of weights is:

$$w_l = 1 + C(N_l + B)^{-A}$$

Where w_l is weight assigned to a label l with N_l users. A and B are application specific parameters and $C = (\log N - 1)(B + 1)^A$. Where N is the observed ground truth dataset size.
Note that weight assigned is decreasing function of N_l hence rare items will get more opportunity of recommendation. But the issue with PfastreXML[(2)] is it takes too much time and space as it uses propensity scored function and other functions different from FastXML[(1)].
Thus we decided to modify our FastXML[(1)] algorithm by simply adapting the above formula from PfastreXML[(2)] and use it to assign weights to the labels in our algorithm. The value of A and B were chosen to be 0.55 and 1.5 resp. as mentioned in the paper of PfastreXML[(2)] itself. Initially we calculated value of C using the above formula and later after experimenting with some nearby values settled on that value which gave the best result. sIn the end, the value of C was chosen to be roughly 14.4.

References

[1] $http://manikvarma.org/pubs/prabhu14.pdf$

[2] $http://manikvarma.org/pubs/jain16.pdf$

[3] $https://www.youtube.com/watch?v=1X71fTx1LKA$