CS771 Assignment-2

Abhishek Soni (170034) **Aniket Gupta (170108)** **Bholeshwar Khurana (170214)**

Saurav Prakash (170646) **Sayak Chakrabarti (170648)**

Abstract

Recommendation systems are a class of information filtering systems that aim to predict the chance (probability) of a user liking an item based on previous experiences of several users with a subset of the items. In this project we aim to implement a simpler version of recommendation system viewing it as a *multiclass classification* problem. Despite the simplicity it is still quite powerful and has a wide range of use in ecommerce, social media, internet searches and other sectors.

Introduction:

In this recommendation system we consider data collected on users over their choices of items. We consider each user as a d dimensional feature vector $\mathbf{x}^i \in \mathbb{R}^d$. The feature vectors are, on average, very sparse, a very less proportion of these coordinates are non-zero. The values contain information obtained from user buying/using the item in the past.
The total number of items in the inventory is denoted by L. For each user i there is a vector $y^i \in \{0,1\}^L$ which indicates if that user likes the items or not. j^{th} coordinate of y^i as 1 means that the user likes the item, 0 means that the user either dislikes it or has not interacted with the item. It is however possible that the user might like multiple items and y^i has multiple coordinates as 1.
The goal of this project is, given an user with his/her feature vector, the recommendation system outputs the top 5 labels in which the user will be most interested, i.e. the items with predicted probabilities closest to one.

Literature:

For training of the recommendation system we were given with $n = 10000$ users with the feature vector of each user, x^i, being of dimension $d = 16385$. The feature vectors were sparse and the average number of non-zero features in a data point is only $\hat{d} \approx 519$. The total number of items were $L = 3400$ and each user was associated with a label vector $i \in \{0,1\}^L$.
Two types of performance measures used were:

1. Precision at k (prec@k): For this, some value of $k \in [5]$ was chosen. Then we check for what fraction of the k most probable items suggested by the recommendation system were actually liked by the user. This will be a fraction $\in [0,1]$. The average of this fraction over all the users denote the prec@k.
2. Macro precision at k (mprec@k): In this, first we chose some $k \in [5]$.Then we run over each of the items and look at all the test users who liked them. The fraction of users for whom the first k recommendations included that item was calculated and the average of this fraction for all the users is the value of mprec@k.

Preprint. Under review.

1 Method Used:

The algorithm proposed by Jain et al., KDD 2016 in PFastreXML[1] was modified and used to train the recommendation system. This is an extreme classification technique using decision trees to give recommendation to users.
The algorithm for PFastreXML focusses on improving the loss function which is crucial for the training algorithm to learn a better model. This project also includes tuning the hyperparameters for better learning as well as experimenting on minor changes such introducing a newer kernel and slightly changing the loss function. The loss function used in this focusses on prioritizing the few relevant labels over the thousands of irrelevant ones. This includes ranking different labels.
It was proved in Jain et al. 2016 that the proposed propensity scored losses computed on the observed labels provided unbiased estimates of the correct loss function computed over the complete dataset without any missing labels, which is however unobtainable.

We have $\mathbf{y}^*, \mathbf{y} \in \{0,1\}^L$ which denote the complete (but unobserved) and observed (and with missing labels) ground truth label vectors for a given data point such that $\mathbf{y}^* = \mathbf{y} = 1$ for observed and relevant items, $\mathbf{y}^* = 1\mathbf{y} = 0$ for unobserved and relevant items, and $\mathbf{y}^* = \mathbf{y} = 0$ for irrelevant items.
The propensity is defined as $p_{il} = P(y_{il} = 1 | y^*_{il} = 1)$ which denotes the marginal probability of a relevant label l being observed for a data point i. There is no contraint placed on the propensities except that $P(y_{il} = 1 | y^*_{il} = 0) = 0$. This means that labels cannot go missing independently or uniformly at random.
The propensity scored loss function is denoted by $\mathcal{L}^*(\mathbf{y}^*, \hat{\mathbf{y}})$. We have $\mathcal{L}^*(\mathbf{y}^*, \hat{\mathbf{y}}) = \sum_{l=1}^{L} \mathcal{L}^*_l(y^*_l, \hat{y}_l) = \sum_{l:y_l=1}^{L} \mathcal{L}^*_l(1, \hat{y}_l)$ that denote the class of loss functions that are calculated solely over relevant items, i.e., $\{l | y_l = 1\}$. $\mathcal{L}^*$ is the true loss function for estimating $\hat{y}$ when the complete ground truth vector y^* is available, which is however not so. Hence we calculate the propensity score on the observed ground truth vector y as $\mathcal{L}^*(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{l=1}^{L} \mathcal{L}^*_l(y_l, \hat{y}_l) = \sum_{l:y_l=1}^{L} \mathcal{L}^*_l(1, \hat{y}_l)/p_l$. It has been proved in the paper that $\mathcal{L}$ is a viable proxy for $\mathcal{L}^*$ and can be used for training and further hyperparameter tuning and performance evaluation.

Based on empirical observations and results obtained, we have used the propensity as a sigmoid function of $\log N_l$ as

$$p_l = P(y_{il} = 1 | y^*_{il} = 1) = \frac{1}{1 + Ce^{-A \log(N_l + B)}} \tag{1}$$

where N_l is the number of data points annotated with label l in the observed ground truth dataset of size N. A, B are application specific parameters and and $C = (\log N - 1)(B + 1)^A$. In our model we have used $A = 0.8, B = 2.5$.
It is assumed that each label can be predicted independently based on a hyperspherical decision boundary generated by

$$P(y^*_{il} | \mathbf{x}_i) = \frac{1}{1 + v_{il}^{2y^*_{il} - 1}} \text{ where } v_{il} = e^{\frac{\gamma}{2} \|\mathbf{x}_i - \mu_l\|_2^2} \tag{2}$$

Optimization: The MLE estimation is carried out as

$$\mu^*_l = \prod_{i=1}^{N} \prod_{l=1}^{L} \sum_{y^*_{il}=0}^{1} P(y_{il} | y^*_{il}) P(y^*_{il} | \mathbf{x}_i)$$

which is same as

$$\mu^*_l = \arg\max_{\mu_l} \sum_{i=1}^{N} \log((1 - y_{il}) + \frac{p_{il}(2y_{il} - 1)}{1 + v_{il}}) \tag{3}$$

Optimizing this using first order derivative yeilds $\mu^*_l = \frac{\sum_{i=1}^{N} u_{il} x_i}{\sum_{i=1}^{N}}$ where $u_{il} = \frac{v_{il}}{1+v_{il}} - \frac{(1-y_{il})v_{il}}{1+v_{il}-p_{il}}$.
For a dataset of high dimensions we usually have $\frac{\gamma}{2}\|\mathbf{x}_i - \boldsymbol{\mu}_i\| >> 0 \forall i \in [N]$ which lead to simplification as $u_{il} \approx y_{il}$ yeilding

$$\boldsymbol{\mu}^*_{\boldsymbol{l}} = \frac{\sum_{i=1}^{N} y_{il} \mathbf{x}_i}{\sum_{i=1}^{N} y_{il}} \tag{4}$$

This can be interpreted as mean of all the datapoints that were labelled to be relevant.
With these equations we proceed to find a model for the recommendation system.
Overall PFastreXML gave better results compared to the other methods. So we decided to work on it and try to improve it further.

2 Advantages and Disadvantages:

We found the following advantages and disadvantages of PFastreXML:

2.1 Advantages:

- It annotates each data point with the most relevant subset of labels from an extremely large label set.
- It prioritizes predicting the few relevant labels over the large number of irrelevant ones.
- It does not erroneously treat missing labels as irrelevant but instead provides unbiased estimates of the true loss function even when ground truth labels go missing under arbitrary probabilistic label noise models.
- It promotes the accurate prediction of infrequently occurring, hard to predict, but rewarding tail labels.
- The algorithm efficiently scales to large datasets with up to 9 million labels, 70 million points and 2 million dimensions giving significant improvements over the state-of-the-art.

2.2 Disadvantages:

- It has a larger model size compared to newer tree-based algorithms.
- It has longer training time compared to newer tree-based algorithms.
- The splitting of nodes uses nDCG Loss function which might not be suitable for some other datasets.*

* However for the given dataset, this nDCG Loss function produces better results as compared to other algorithms.

3 Experiment:

The source code of PFastreXML was downloaded from [2]. The model was trained on the given dataset of 10000 users and was again tested on the same training set. The hyper-parameters used (after tuning) were $A = 0.8, B = 2.5, \alpha = 0.8, \gamma = 30$. The following results were obtained:

k	prec@k	mprec@k
1	91.0%	6.84%
3	83.0%	18.32%
5	75.1%	28.97%

Table 1: Results

4 Modifications:

4.1 PFastreXML:

PFastreXML uses inverse propensity loss function. It is based on giving priorities to items that are used less frequently as not much is known about them. PfastreXML optimizes nDCG function which is $\Sigma_{l=1}^{k} \frac{y_{r_l}}{\log(1+l)}$ we tried modifying this optimising function to $\Sigma_{l=1}^{k} \frac{y_{r_l}}{\sqrt{l+1}}$ and got good results although the results were not as good as those obtained with the actual nDCG function.

k	prec@k	mprec@k
1	83.0%	1.24%
3	67.6%	3.25%
5	57.1%	5.9%

Table 2: Results for root func.

We also tried modifying it to $\Sigma_{l=1}^{k} \frac{y_{r_l}}{(l+1)^{1/4}}$. and the results obtained were:

k	prec@k	mprec@k
1	83.4%	1.0%
3	67.9%	3.25%
5	57.3%	5.86%

Table 3: Results for pow(1/4) func.

k	prec@k	mprec@k
1	84.4%	1.46%
3	68.6%	4.34%
5	56.3%	6.53%

Table 4: Results for logarithmic func.

The above results are for held-out validation carried on 70-30 random split of the training data and then their hyperparameters were tuned.

4.2 Parabel:

Before switching to PFastreXML we worked on Parabel[3], where we tried several improvements over the actual algorithm. Such as we tried modifying the similarity function of labels used by it from linear kernels to quadratic , gaussian kernels, we also tried using k++ initialization for the kmeans label splitting at nodes. We even tried imbalancing the tree such that we allow any number of lables from $k/4$ to $3k/4$ in each node(such that we were splitting the sorted array as soon as we fall down below the mean in the interval). But the results were even worse than the simple Parabel. But the results of this method were not as good as the original PFastreXML implementation. Some of the results are as shown:

k	prec@k	mprec@k
1	48.8%	1.23%
3	49.6%	3.55%
5	46.7%	5.5%

Table 5: Results for Quadratic kernel.

We did not use Parabel because PFastreXML was better than Parabel in terms of both prec as well as mprec.

k	prec@k	mprec@k
1	84.4%	0.5%
3	68.6%	1.5%
5	57.43%	3.46%

Table 6: Results for linear kernel

k	prec@k	mprec@k
1	83.3%	0.496%
3	65.4%	1.213%
5	55.23%	3.27%

Table 7: Results for imbalanced trees

References

[1] H. Jain, Y. Prabhu, and M. Varma. 2016. PFastreXML: Extreme Multi-label Loss Functions for Recommendation, Tagging, Ranking 38; Other Missing Label Applications. In KDD.

[2] Code for PfastreXML. http://manikvarma.org/code/PfastreXML/download.html

[3] Prabhu, Y., Kag, A., Harsola, S., Agrawal, R., Varma, M.: Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In: Proceedings of the 2018 World Wide Web Conference on World Wide Web. pp. 993–1002 (2018)