Project 2

Project Title : Recommendation System

Task 1

From [3], we used the method of FastXML to create our recommendation system

Few Salient Features of FastXML:

1. FastXML learns a hierarchy over the feature space instead of label space. Very few labels are active in each sector of feature space.

2. Prediction is made by identifying the part where a point resides by going through the acquired space hierarchy.

3. Tree methods are more suitable for extreme multi-label classification problems.

4. The approach to ranking and recommendation task involves learning a function that maps a function to a subset of relevant items from a large item set and then given new user would use this learnt function to get recommended a set of items relevant to the user.

5. FastXML is a tree based method which uses novel rank-sensitive node partitioning objective function based on nDCG. This node partitioning function gets more balanced prediction accuracy. This is optimised via alternating optimization. We learn an ensemble of such hierarchy generating trees. Each tree learns a different hierarchy as compared to others.

6. When a test instance comes in, it is dropped down each of these trees down to the leaf nodes. the leaf nodes contain a label probability distribution over small set of labels which are active in them. To make final predictions we aggregate the probability distributions over all the different trees and rank the items based on the final probability scores. Objective directly minimises the ranking loss which ensures a good ranking. Explanation of Node Splitting Mechanism in feature space :

In the question, each user is associated with a set of items they like. The union of all the sets comprises the active item sets in the node. The node partitioning algorithm begins with the random allotment of users to left or right child with equal probability.
δ_i are binary variables taking values +1 and -1 indicating which child the i-th instance belongs to. In the next step, we rank the labels in the left child according to the frequency of occurrence among the instances of the left child. Similarly we rank the labels in the right child. In practice, instances associated with the large number of labels tend to have larger influence than ones with fewer labels, hence we normalise the contribution of different instances using weights N_y as shown in the equation

$$r^{\pm *} = rank(\sum_{i:\delta_i = \pm 1} N_{y_i} y_i)$$

In the next step we re-allot each instance to the node whose ranking gives the instance the higher nDGC gain.

$$\delta_i^* = sign(v_i^- - v_i^+) \forall i,$$

where

$$v_i^{\pm} = -C_r \sum_j \frac{N_{y_i} y_{ij}}{\log(1 + r_j^{\pm})}$$

Preprint. Under review.

In the final step of node splitting we will learn a linear separator in the feature space by solving a L1 regularize logistic regression given below

$$\min_{w} \|w\|_1 + \sum_{i} C_{\delta}(\delta_i) log(1 + \exp^{-\delta_i w_t x_i})$$

The L1 regularizer helps in avoiding overfitting and also creating more compact models. Node partitioning procedure is applied recursively to the left child and then to right child node.

Now for each new test user, we travel the ensemble of trees and calculate the average probability of labels which are active in the leaf nodes to which the new user goes to and hence in this way the top five items with maximum probability is recommended to the user.

Task 2

Advantages

1. We use Decision Trees in FastXml, so it is good in prediction as compared to methods which use One vs All, etc.

2. FastXML is faster to train and it makes more accurate predictions as compared to the state-of-the-art MLRF(Multi-label Random Forest) and LPSR(Label Partitioning by Sublinear Ranking). FastXML can be significantly faster than MLRF since the proposed node partitioning formulation in FastXML can be optimized more efficiently than the entropy or Gini index based formulations in MLRF. Also, FastXML can be faster to train than LPSR which has to first learn computationally expensive base classifiers for accurate prediction.

3. FastXML's node partitioning formulation optimizes a rank sensitive loss which can lead to more accurate predictions over MLRF's Gini index or LPSR's clustering error.

4. FastXML could efficiently scale to problems involving a million labels where accurate training of MLRF, LPSR and 1-vs-ALL models would require a very large cluster.

Disadvantages

1. FastXML needs a lot of memory as it trains an ensemble of trees so it has to store all the training data for prediction.

2. FastXML which uses Decision Trees, takes more time in training as compared to all the methods which use OVA.

3. Tree classifiers are still prone to predicting tail labels with low probabilities because of partitioning errors in the internal nodes.

Task 3

We have used the code from the repository [2] and after training them and testing them on the same data with default hyperparameters given in the code which was downloaded from the repository, we get the results:

prec@1	0.872
prec@3	0.774
prec@5	0.679
mprec@1	1.589e-03
mprec@3	1.222e-02
mprec@5	3.117e-02

Task 4

We chosed FastXML [3] algorithm implementation from the XML code repository to solve the multi-label classification problem. The prediction file of python at hand had the feature vectors in scipy

CSR matrix format and our C++ code required SVM lite data format. Thus a file format dumping method had to be implemented, where the standard scipy dump_svmlight_file method contributed lot to the prediction time. Thus we followed the dump svmlight method in svmlight_loader package [1] available online to implement the conversion which reduced the prediction time a lot.

We tried tuning the hyperparameters for the above approach. The default hyperparameters of the method were Starting tree (s=0), Number of trees (t=50), Feature bias value (b=1), SVM weight co-efficient (c=1), Maximum number of instances allowed in a leaf node (m=10), Number of label probability pairs to retain in a leaf(l=10).

For the default values of the hyperparameters the evaluation measures obtained were:

Prec@1	Prec@3	Prec@5	mPrec@1	mPrec@3	mPrec@5	Model size
0.831	0.672	0.572	1.121e-3	7.485e-3	1.681e-2	31MB

1. Starting tree (s):

Values of s	Prec@1	Prec@3	Prec@5	mPrec@1	mPrec@3	mPrec@5	Model size
5	0.83	0.671	0.572	1.11e-3	7.513e-3	1.693e-2	34.1MB
25	0.831	0.674	0.571	1.082e-3	7.569e-3	1.709e-2	46.5MB

2. Number of trees (t):

Values of t	Prec@1	Prec@3	Prec@5	mPrec@1	mPrec@3	mPrec@5	Model size
25	0.828	0.672	0.571	1.137e-3	7.551e-3	1.701e-2	15.5MB
40	0.83	0.671	0.573	1.119e-3	7.575e-3	1.701e-2	24.8MB
60	0.829	0.672	0.572	1.1040e-3	7.551e-3	1.698e-2	37.2MB

3. Feature bias value (b):

Values of b	Prec@1	Prec@3	Prec@5	mPrec@1	mPrec@3	mPrec@5	Model size
0	0.832	0.677	0.572	1.071e-3	7.75e-3	1.672e-2	30.7MB
0.5	0.828	0.672	0.572	1.041e-3	7.508e-3	1.681e-2	30.7MB
2	0.83	0.672	0.574	1.094e-3	7.527e-3	1.654e-2	31.9MB

4. SVM weight co-efficient (c):

Values of c	Prec@1	Prec@3	Prec@5	mPrec@1	mPrec@3	mPrec@5	Model size
2	0.833	0.679	0.574	1.351e-3	7.961e-3	1.781e-2	54.7MB

5. Maximum number of instances allowed in a leaf node (m):

Values of m	Prec@1	Prec@3	Prec@5	mPrec@1	mPrec@3	mPrec@5	Model size
5	0.829	0.674	0.571	1.09e-3	7.468e-3	1.705e-2	31.6MB
12	0.829	0.674	0.571	1.097e-3	7.445e-3	1.636e-2	30.7MB
15	0.828	0.671	0.571	1.079e-3	7.369e-3	1.624e-2	31.2MB

6. Number of label probability pairs to retain in a leaf(l):

Values of l	Prec@1	Prec@3	Prec@5	mPrec@1	mPrec@3	mPrec@5	Model size
5	0.828	0.677	0.575	1.101e-3	8.198e-3	1.953e-2	29.2MB

After much tuning, we think the model performs best with default hyperparameters as tuning improved precision but had a significant tradeoff with other metrics like model size, etc.

References

[1] Lars Buitinck Mathieu Blondel. *svmlight-loader*. URL: https://github.com/mblondel/svmlight-loader.

[2] Yashoteja Prabhu and Manik Varma. *FastXML*. URL: http://manikvarma.org/code/FastXML/download.html.

[3] Yashoteja Prabhu and Manik Varma. "FastXML: A Fast, Accurate and Stable Tree-classifier for Extreme Multi-label Learning". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA:

ACM, 2014, pp. 263–272. ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623651. URL: http://doi.acm.org/10.1145/2623330.2623651.