CS771 : Introduction to Machine Learning

Assignment 2

2nd November 2019

Group 19

Siddharth Jayashankar (170699) Keshav Bansal (170335)

Harsh Agrawal (170287) Snehal Raj (170705)

Durgesh Rajendra Agrawal (170262)

Part 1

Question

Take one reference from the following list (references [2, 5, 31, 32, 33, 34, 36, 40, 41] in the repository), give a citation to that paper and briefly describe the method in your own words. You do not have to give all minute algorithmic details (since we can always just read the paper itself) but do explain the salient points which are crucial to the algorithm.

Solution

Method Chosen : **PFastreXML**: *Extreme Multi-label Loss Functions for Recommendation, Tagging, Ranking & Other Missing Label Applications* [1]

Algorithm Details:

- PfastreXML is an improved version of FastXML, made by replacing the nDCG loss with its propensity scored variant and using additional classifiers designed for tail labels. PfastreXML optimizes propensity scored nDCG by leveraging FastXML for nDCG optimization. PfastreXML then further extends FastXML to improve tail label prediction which is the most challenging aspect of extreme multi-label learning. PfastreXML achieves this at scale by making key approximations which increase FastXML's training time by just seconds while retaining the prediction accuracy gains of the extension.
- Learning the PFastreXML classifier primarily involves learning two components,
  1. **FastXML classifier** - FastXML learns an ensemble of trees during training. Trees are grown by recursively partitioning nodes starting at the root until each tree is fully grown. Nodes are split by learning a separating hyperplane which partitions training points between a left and a right child. A node N is split by learning a hyperplane and bias to partition training points between its left and right children. Node partitioning terminates when a node contains fewer points than a threshold. Leaf nodes contain a probability distribution over the label set.
  2. A re-ranker which attempts to recover the tail labels missed by PFastXML. The re-ranker is essentially a **Rocchio classifier** which is a nearest centroid classifier that assigns to observations the label of the class of training samples whose mean (centroid) is closest to the observation. Thus the classifier assigns the test instance to the label with closest centroid among the top 1000 labels predicted by PFastXML. [4]
- Propensity scored variant of nDCG loss is unbiased and assigns higher rewards for accurate tail label predictions. Propensity scored losses
  1. Prioritize predicting the few relevant labels over the large number of irrelevant ones;
  2. Don't erroneously treat missing labels as irrelevant but instead provide unbiased estimates of the true loss function even when ground truth labels go missing under arbitrary probabilistic label noise models; and

3. Promote the accurate prediction of infrequently occurring, hard to predict, but rewarding tail labels.

- The final scores assigned to label for a test instance is given by a convex combination of scores PFastXML and the Rocchio classifier for top 1000 labels. Although the data set supplied has 3400 labels, running the Rocchio classifier on the top 1000 labels predicted by FastXML was sufficient as not many labels are relevant to all users. This also enabled us to reduce the prediction time. [1]

Part 2

Question

Discuss a few advantages and disadvantages of the method you just described above.

Solution

- *Advantages*
  1. PfastreXML results in better classification of tail labels (labels which are infrequent and hard to predict). The dominance of PfastreXML is partly explained by the fact that it is trained with label propensities that are computed with additional external information.
  2. FastXML is an important component of PfastreXML, which optimizes the normalized Discounted Cumulative Gain (nDCG). nDCG is a measure which is sensitive to both ranking and relevance and therefore ensures that the relevant positive labels are predicted with ranks that are as high as possible (which cannot be guaranteed by rank insensitive measures such as Gini index). [2]
  3. PfastreXML is a tree based approach, hence the prediction time is much better than other approaches like LWP.
- *Disadvantages*
  1. The algorithm involves large number of hyper parameters and large model size.
  2. Performance of PFastreXML depends heavily on the good performance of Rocchio_{1000} classifier, which in turn is learnt from the top labels predicted by PFastXML classifier.
  3. Metadata may not always be available to measure propensity. [4]

Part 3

Question

Download an implementation of the method you chose (or if you wish, implement yourself). Train the method you chose on the train data we have provided you and test it on the train data itself to give us the following statistics: prec@k and mprec@k for k = 1, 3, 5.

Solution

A	B	γ	Data Used	prec@1	mprec@1	prec@3	mprec@3	prec@5	mprec@5
0.55	**1.5**	**30**	**80-20 Train-Test Split**	0.82	0.067	0.56	0.013	0.036	0.056
			Train and Test over entire data	0.92	0.075	0.84	0.18	0.77	0.3

Part 4

Question

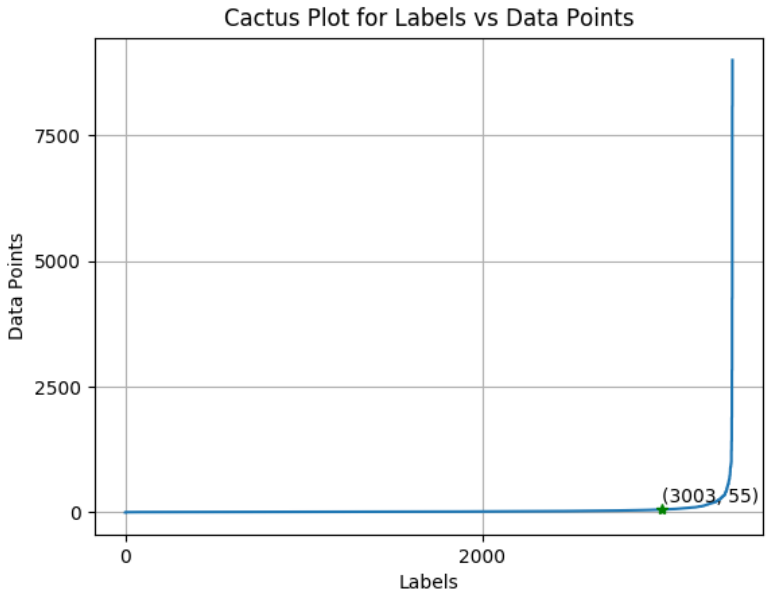
Suggest a method that you would use to solve the problem.

Solution

We have chosen PfastreXML to solve the problem.

To check if the suggested model would work well with the provided dataset, we checked the variation in number of data points with the number of labels. As shown in the graph below, the data given to us contains a lot of tail labels. Out of 3400 labels, around 3000 have count less than 55, whereas the maximum count for a particular label is 8082. This strengthened our hypothesis for choosing PfastreXML as it has a proven advantage in presence of tail labels.

Figure 1: Data Points vs Number of Labels



The inverse propensity model used in the algorithm requires setting hyper parameters A and B, which depend on the metadata. As mentioned in the disadvantages section, the metadata for choosing the best parameter values may not always be present. Since we did not have any metadata for this dataset, we used held-out validation with an eighty-twenty train-test split and found the best values for A and B. The same technique was used to train other hyper parameters.

We first tried out different values of each hyper-parameters to get an idea of a range where we should search for the best values of hyper parameters. From the results we obtained, we saw that not all hyper parameters affected the precision and macro precision equally and thus we decided to tune a subset of the hyper params leaving the rest to their default values. The hyper parameters that we found to be causing the most impact were the parameters A and B, number of trees, gamma parameter in tail label classifier and the number of max-leaf instances allowed in the leaf node.

After carefully analysing the trends and variations, we settled on a range of values for hyper parameter tuning and then tuned the hyper parametres using our train-test splits. The best values

found by us are as mentioned below.

A	B	γ	prec@1	mprec@1	prec@3	mprec@3	prec@5	mprec@5
0.8	**2.0**	**170**	0.78	0.027	0.66	0.062	0.57	0.085
0.7	**2.0**	**185**	0.79	0.032	0.64	0.069	0.55	0.093
0.55	**1.5**	**120**	0.79	0.025	0.65	0.058	0.57	0.082

Table 1: Results for different hyperparameters for 80-20 Train-Test Split (# trees = 25)

By tuning our hyperparametres, we were able to increase the precision and macro-precision values by approximately 10%.

The getReco function in *predict.py* takes a sparse matrix as an argument, but our PFastre prediction executable takes matrix as file. After careful analysis of the code being used, we found that the main bottleneck in our prediction times was writing to and reading from the files.

Therefore instead of using the function *dumpData()* in utils.py, we use the *tocoo* function provided in scipy which is used to return the coordinate representation of a scipy sparse matrix. This reduced the time taken to dump the matrix into a file and in turn our prediction time reduced considerably.

As mentioned in the disadvantages, the PFastreXML method has a relatively large model size. Initially, the model size was 200 MB for 25 trees and model w. To reduce the model size, we stored the model file w in binary and modified the *pfastre_predict* to read from this binary files. This reduced the size to approximately 70MB.

Part 5

Question

Train your chosen method on the train data we have provided (using any validation technique you feel is good). Store the model you obtained in the form of any number of binary/text/pickled/compressed files as is convenient and write a prediction method in Python in the file *predict.py* that can take a new data point and use the model files you have stored to make predictions on that data point. Include all the model files, the *predict.py* file, as well as any library files that may be required to run the prediction code, in your ZIP submission. You are allowed to have subdirectories within the archive.

Solution

The code from [3] has been used with appropriate hyper-parameter tuning and some minor modifications.

References

[1] Himanshu Jain, Yashoteja Prabhu, and Manik Varma. 2016. Extreme Multi-label Loss Functions for Recommendation, Tagging, Ranking & Other Missing Label Applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM.

[2] Y. Wang, L. Wang, Y. Li, D. He, and T.-Y. Liu. A theoretical analysis of nDCG type ranking measures. In *COLT*, pages 25–54, 2013.

[3] Manik Varma, The Extreme Classification Repository: Multi-label Datasets & Code, http://manikvarma.org/downloads/XC/XMLRepository.html.

[4] R. Babbar and B. Schölkopf, "Adversarial extreme multi-label classification," 2018, *arXiv:1803.01570*.