Extreme Classification using Autoencoders

Vishwak Srinivasan
CS15BTECH11043

Sukrut Rao
CS15BTECH11036

Harsh Agarwal
CS15BTECH11019

ABSTRACT

The problem of extreme classification requires selecting a subset of labels for each data point among a large set of labels. Using conventional multi-label classification techniques could be challenging and computationally inefficient when the label space is high dimensional. In this project, we use autoencoders for dimensionality reduction, and propose two approaches: `NeuralXC` and `HierarchicalXC`. In `NeuralXC`, we use autoencoders for encoding both input feature vectors and output label vectors in lower dimensional space, and then perform regression. In `HierarchicalXC`, we create a tree of classifiers based on hierarchical structure of the labels, and pass an encoded input to obtain the labels. We provide preliminary results and outline future work.

Keywords

Extreme Classification, Autoencoders, Multi-label Classification

1. INTRODUCTION AND MOTIVATION

Extreme Multi-label Classification (XC) (also known as Extreme Classification) is the task of learning a classifier which can assign a small subset of relevant labels to an instance from an extremely large subset of target labels. An example would the task of tagging Wikipedia articles. One might wish to build a classifier that annotates a new article or web page with the subset of most relevant Wikipedia categories which are in millions.

XC is different from normal classification since XC predicts a subset of labels as opposed to a single label. We will be able to create an instance of normal classification from XC by classify one instance from all possible subsets of classes there could be for a data-point, but this is extremely inefficient. Considering the same example as above, a normal classification task would have to classify one target out of 2^{10^6} possible tasks, which would be intractable and less-confident as well.

CS6370 - Information Retrieval Spring 2018, IIT Hyderabad

ACM ISBN .

DOI:

Our motivation for using auto-encoders is due to two reasons:

- Features are extremely high dimensional, and traditional algorithms face difficulties dealing with high dimensional data - otherwise known as "curse of dimensionality".
- Lack of efficient algorithms to deal with sparse vectors arising from multi-hot encodings of outputs.

The rest of the report is organized as follows: Section 2 discusses previous approaches to this problem and Sections 3 and 4 discuss `NeuralXC` and `HierarchicalXC`- our two approaches. Section 5 discusses our experimental setup and results.

2. RELATED WORK

A lot of progress has been made in developing algorithms for extreme classification tasks. Most of these algorithms can be categorized into 4 different classes: **One-vs-All approaches**, **Tree-based approaches**, **Embedding based approaches** and **Deep learning methods**. The methods that we propose fall under the deep learning methods (`NeuralXC`) and tree based approaches (`HierarchicalXC`).

2.1 One-vs-All approaches

One-vs-All approaches include methods like Parabel[10], DisMEC[1], Max-margin Multi-label classification (M3L)[3], PD-Sparse[16] and PPD-sparse[15]. All of these models perform reasonably well, and have low model sizes as well. However, for most of these models the training process could be extremely high due to requirement that each classifier would have to train data-points not relevant for the label. PPDSparse[15] tries to resolve this issue to a limited extent by optimizing over a shortlist of negative training examples based on a primal and dual sparsity preserving algorithm, which causes it to be 100x faster than the state-of-the-art DisMEC[1] on certain datasets, while providing similar accuracy. Parabel[10] is faster than these methods by up to 600x. This is achieved by subsampling data-points based on the intuition that accuracy can be maintained while restricting each label's negative training examples to those points annotated with the most similar, or confusing, labels - thereby achieving a balanced label hierarchy.

2.2 Tree-based approaches

Tree approaches to extreme classification are extremely efficient training-wise and therefore have low training and

prediction times but have high model sizes and poor prediction accuracy. Leading approaches such as [9], [4] and [11] learn a large ensemble to compensate for the poor prediction accuracy of any single tree by using weak constant classifiers in their leaf nodes. Probabilistic Label Trees[5] optimizes directly over the F-measure and provide sparse probability estimate using which they provide the subset of label that are most likely to occur for a given data-point.

2.3 Embedding-based approaches

SLEEC[2] is a formulation for learning a small ensemble of local distance preserving embeddings which can accurately predict infrequently occurring (tail) labels. This allows SLEEC[2] to boost classification accuracy by learning embeddings that preserve pairwise distances between only the nearest label vectors.

Another popular approach based on embeddings is AnnexML[13]. At training step, AnnexML[13] constructs k-nearest neighbor graph of the label vectors and attempts to reproduce the graph structure in the embedding space. The prediction is efficiently performed by using an approximate nearest neighbor search method which efficiently explores the learned k-nearest neighbor graph in the embedding space.

2.4 Deep learning methods

XML-CNN[7] combines the strengths of existing CNN models and goes beyond by taking multi-label co-occurrence patterns into account in both the optimization objective and the design of the neural network architecture, and scales successfully to the largest extreme multi-label text classification benchmark datasets.

3. NEURALXC

`NeuralXC` is a deep learning architecture for efficiently learning the problem of extreme multi-label classification. This approach is on training multiple regression problems in a low-dimensional space induced by the encodings of the features and the multi-hot encoded vectors of the outputs.

The approach is detailed as follows:

- We train an auto-encoder over the training features to reduce the dimensionality of the these feature vectors and obtain low-dimensional representations.
- We also train an auto-encoder over the multi-hot encoded vectors to produce a dense low-dimensional embedding of these multi-hot encoded vectors. A *multi-hot encoded vector* is a binary vector consisting of 0s and 1s - 1 where the class label is relevant and 0 where the class label is irrelevant. Naturally this vectors would be extremely sparse in expectation, and the auto-encoding helps us generate a dense low-dimensional embedding / representation of this sparse vector.
- We then perform a regression task over these low dimensional representations over inputs an outputs.

By adopting this approach, we have converted the problem of extreme multi-label classification to three regression problems.

Some preliminary notations and definitions

Denote d_x, d_y as the original dimensionality of the features and multi-hot vector of outputs. Note that $d_y = C$ where C is the number of classes in the problem. Also denote ld_x, ld_y as the dimensionality of the low-dimension representations of the features and outputs.

Define $\phi_i : \mathbb{R}^{d_x} \to \mathbb{R}^{ld_x}$ as the feature / input encoder function. Similarly define $\phi_o : \mathbb{R}^{d_y} \to \mathbb{R}^{ld_y}$ as the output (in multi-hot encoded form) encoder function. Let $\psi_i : \mathbb{R}^{ld_x} \to \mathbb{R}^{d_x}$ as the feature decoder, and $\psi_o : \mathbb{R}^{ld_y} \to \mathbb{R}^{d_y}$ as output decoder. Also define the $f : \mathbb{R}^{ld_x} \to \mathbb{R}^{ld_y}$, which is the regression function. It is expected that the following properties hold:

- $\psi_i(\phi_i(x)) = x$ (Input auto-encoder condition)
- $\psi_o(\phi_o(y)) = y$ (Output auto-encoder condition)
- $\psi_o(f(\phi_i(x))) = y$ if $(x, y) \in \mathcal{D}$ (Regression condition)

3.1 The loss function

Based on the definitions above, we define a loss function to optimize over:

$$L(x,y) = \alpha \underbrace{||\psi_i(\phi_i(x)) - x||_2^2}_{\text{Input Reconstruction}} + \beta \overbrace{||\psi_o(\phi_o(y)) - y||_2^2}^{\text{Output Reconstruction}} + \underbrace{L_{BCE}(\psi_o(f(\phi_i(x))), y)}_{\text{Classification}} \quad (1)$$

L_{BCE} is the binary cross entropy loss computed as follows:

$$L(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \quad (2)$$

where y is the ground truth and $\hat{y}$ is the prediction. Optimization is done using the chain rule and back-propagation. α and β are importance factors to the encoding losses. Theoretically speaking, if the values of the α and β were high, then the encodings would be better, but would compromise the classification task at hand. Similarly, if α and β were low, the classification loss is given more importance, which might lead to bad encodings and decodings of the features and the representations respectively.

The high level architecture is shown in Figure 1.

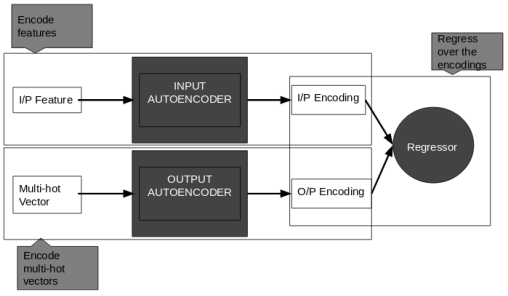

Figure 1: The blackboxes/circles represent neural network models. A forward would involve input encoding, regression and decoding of the regression output.

4. HIERARCHICALXC

Often, classes in the real-world follow a hierarchy. For example, in the biology, one way of classifying all living organisms is based on taxonomic ranks, starting from kingdom

and ending with species. In such a ranking, two species that belong to the same kingdom are said to be closer to each other in the taxonomy than those that do not. Taking forward the analogy to the extreme classification problem, it is possible that label classes are also distributed in such a hierarchy, where some labels are closer to each other than others, and consequently might be harder to distinguish from. If such a hierarchy was known, then given a data point, one could decide first among higher levels of classes, and traverse the hierarchy towards lower levels. This should help reducing the search space, since subtrees of classes rejected in higher levels need not be traversed. With this as the motivation, we try another approach, called `HierarchicalXC`, where we first generate such a hierarchy, by assuming it exists, and then train a tree of classifiers for extreme classification.

The `HierarchicalXC` architecture consists of two components:

1. An autoencoder, to reduce dimensionality of input feature vectors. This is similar to that in `NeuralXC`.

2. A hierarchy of multi-label classifiers in the form of a binary tree. Each classifier decides which of its two subtrees of child classes must be further explored, by performing binary multi-label classification. Based on the output, none, one, or both branches might be traversed. The leaf nodes consist of individual labels.

The use of the autoencoder is identical to that in `NeuralXC`. We now describe the creation of the hierarchy of classes.

4.1 Hierarchy of classes

The motivation behind assuming that the classes have a hierarchy, as already described, is that many real-world classes have them. However, we also expect it to have an additional advantage of simpifying the classification task, by reducing the high dimensional multi-class multi-label classification task to a series of multi-label binary classification tasks. Another motivating factor is that one could have dedicated classifiers for hard-to-distinguish classes, potentially improving performance.

To generate the hierarchy, we first aggregate classes into clusters. For this, we use hierarchical complete-link agglomerative clustering. We use co-occurrence of classes as a heuristic for defining the distance metric. The intuitive idea is that if two classes appear together for a large number of data points, then they might be "close" to each other, and vice-versa. So, we define the distance between two classes i and j as:

$$d_{i,j} = -|\{x | x \in c_i \wedge x \in c_j\}|$$

where x belongs to the set of data points, and $x \in c_i$ if x is labelled with the class i. $|.|$ denotes cardinality of the set. This is a crude metric, and its exact definition would not be very important for agglomerative clustering, which only concerns itself with two closest clusters at each step. It can be further refined by normalizing with the union of the occurrences of the two classes (resulting in negative of IoU), since that would also account for the case when two classes co-occur often simply because they are each labelled for a large number of data points.

The hierarchy works as follows:

1. The root classifier deals with the entire set of classes. It then directs a data point to its child subtrees, each of which deal with a partition of classes. The partition in each subtree corresponds to the last-but-one step in the agglomerative clustering.

2. Each classifier inside the tree handles a subset of classes, and passes each data point to its child subtrees after classification.

3. The leaf nodes consist of individual classes, and on reaching such a node, the data point is labelled with that class.

4.1.1 *Difference compared to One-vs-all*

Having separate classifiers dedicated to a subset of classes is similar to the one-vs-all approach. However, there are crucial differences, some of which we list below:

1. Each classifier partitions classes into two disjoint subsets. This contrasts with one-vs-all, where each classifier answers for the presence and absence of a single class as compared with all others.

2. Each classifier handles only a subset of classes (except the root). On the other hand, in one-vs-all, each classifier handles all possible classes. In `HierarchicalXC`, the classifier needs to learn only between the two subsets, and might be useful particularly in cases when pairs of classes are hard to distinguish. Based on our choice of distance metric, such pairs of classes would occur close to the leaves, which would result in there being individual classifiers dedicated to distinguishing between such pairs.

3. Each classifier gets only a subset of data. A classifier only decides data points that are sent to it. So, even though there would be N classifiers if there are N classes, not all of them might be needed for classifying a given data point, unlike in one-vs-all.

4.1.2 *Time Complexity of classification*

Let the number of classes be N. Then, we have N distinct multi-label classifiers created, and linked in the form of a binary tree. Based on the nature of the data, the time complexity for inference would vary.

Let h be the height of the tree. Then, in the best case, when the tree is perfectly balanced, we have $h = \mathcal{O}(\log N)$. On the other hand, if the tree is highly imbalanced, then we would have $h = \mathcal{O}(N)$. Note that in a perfectly imbalanced tree, each node has one of its children as a leaf, and this degenerate case is very similar (but not identical) to the one-vs-all approach.

A data point may have any number of labels associated with it, out of the available labels. In the best case (for time complexity), a data point may traverse exactly one path in the tree from the root to a leaf, which would require it to be classified by $\mathcal{O}(h)$ classifiers. However, if every label applies to it, then it would need to run through every classifier, i.e., $\mathcal{O}(N)$ classifiers.

4.2 Architecture

The architecture for `HierarchicalXC` can be found in Figure 2.

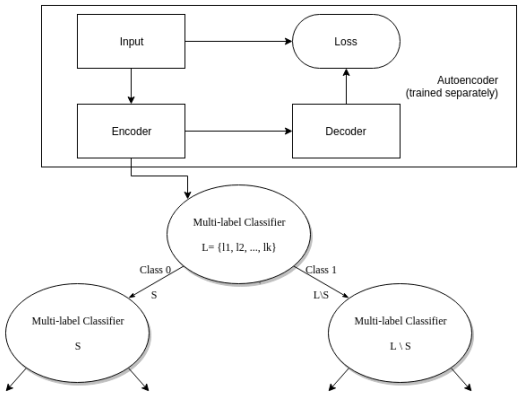

Figure 2: `HierarchicalXC` architecture

5. EXPERIMENTAL EVALUATION

5.1 Datasets

We evaluated `NeuralXC` and `HierarchicalXC` on four datasets: Bibtex[11, 6], Delicious[11, 14], Mediamill[11, 12], and EURLex-4K[2, 8][1].

A summary of the sizes and input and label dimensions of each dataset can be found in Table 1.

Dataset	Feature Dim.	Output Dim.	Training Points	Testing Points
Bibtex	1836	159	4880	2515
Delicious	500	983	12920	3185
Mediamill	120	101	30993	12914
Eurlex	5000	3993	15539	3809

Table 1: Dataset characteristics

5.2 Evaluation Metrics

We use two popular evaluation metrics: Precision @ K (P @ K) and Normalized Discounted Cumulative Gain @ K (NDCG @ K). They are defined as follows:

$$P@k = \frac{1}{k} \sum_{l \in \text{rank}_k(\hat{y})} y_l \tag{3}$$

$$nDCG@k = \frac{DCG@k}{\sum_{l=1}^{\min(k, ||y||_0)} \frac{1}{\log(l+1)}} \tag{4}$$

5.3 Implementation

We implemented our methods in Python. The code can be found at `https://github.com/vishwakftw/extreme-classification`.

5.4 Results

We present preliminary results. A model providing random predictions for each label with probability $\frac{1}{2}$ was used as the baseline.

[1]Datasets can be downloaded from http://manikvarma.org/downloads/XC/XMLRepository.html

5.4.1 `NeuralXC`

The results for `NeuralXC` can be found in Figures 3, 4, 5, and 6.

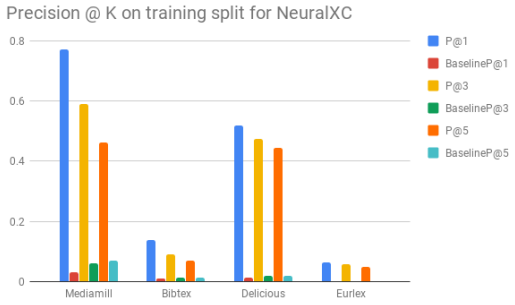

Figure 3: Precision@K on the training split for `NeuralXC`

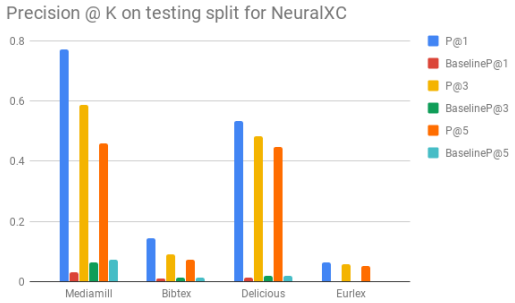

Figure 4: Precision@K on the test split for `NeuralXC`

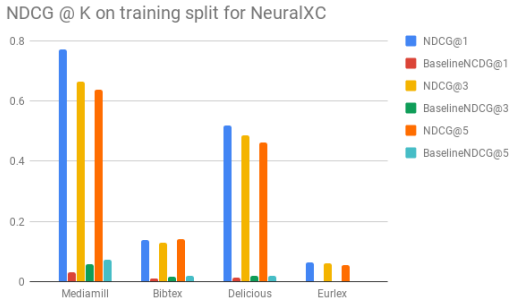

Figure 5: NDCG@K on the training split for `NeuralXC`

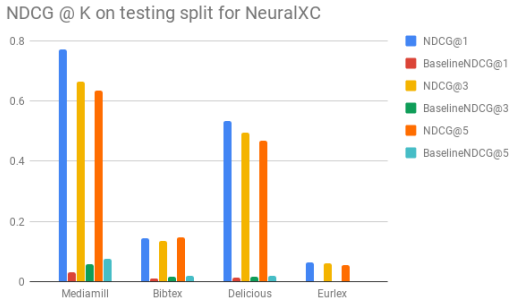

Figure 6: NDCG@K on the test split for `NeuralXC`

5.4.2 `HierarchicalXC`

The results obtained by `HierarchicalXC` were found to be worse than the baseline model. Here, we present results on two of the datasets in Table 2. The idea behind `HierarchicalXC` needs further investigation.

Dataset	Precision @ 5		NDCG @ 5	
	Train	Test	Train	Test
Bibtex	0.0396	0.0410	0.0266	0.0269
Mediamill	0.0102	0.0100	0.0112	0.0111

Table 2: Results for `HierarchicalXC`

6. FUTURE WORK

In this work we have worked on features - perhaps TF-IDF values - generated in a process agnostic to the method of classification. An improvement would be to incorporate a text / language model for feature generation linked to these models inherently. This would give rise to a pipeline where text is fed in its natural form, and we get the corresponding subset for this input.

Another improvement could be involve use of popular language models like Word2Vec to generate these embeddings for us, albeit dense in nature. Note that almost all of the embeddings in this case were sparse, leading to higher dimensions unnecessarily.

7. REFERENCES

[1] R. Babbar and B. Schölkopf. Dismec: Distributed sparse machines for extreme multi-label classification. 2017.

[2] K. Bhatia, H. Jain, P. Kar, M. Varma, and P. Jain. Sparse local embeddings for extreme multi-label classification. 2015.

[3] B. Hariharan, L. Zelnik-Manor, S. V. N. Vishwanathan, and M. Varma. Large scale max-margin multi-label classification with priors. 2010.

[4] H. Jain, Y. Prabhu, and M. Varma. Extreme multi-label loss functions for recommendation, tagging, ranking and other missing label applications. 2016.

[5] K. Jasinska, K. Dembczynski, R. Busa-Fekete, K. Pfannschmidt, T. Klerx, and E. Hullermeier. Extreme f-measure maximization using sparse probability estimates. 2016.

[6] I. Katakis, G. Tsoumakas, and I. Vlahavas. Multilabel text classification for automated tag suggestion. 2008.

[7] J. Liu, W. Chang, Y. Wu, and Y. Yang. Deep learning for extreme multi-label text classification. 2017.

[8] E. L. Mencia and J. Fürnkranz. Efficient pairwise multilabel classification for large-scale problems in the legal domain. 2008.

[9] Y. Prabhu, A. Kag, S. Gopinath, K. Dahiya, S. Harsola, R. Agrawal, and M. Varma. Extreme multi-label learning with label features for warm-start tagging, ranking and recommendation. 2018.

[10] Y. Prabhu, A. Kag, S. Harsola, R. Agrawal, and M. Varma. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. 2018.

[11] Y. Prabhu and M. Varma. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. 2014.

[12] C. G. Snoek, M. Worring, J. C. Van Gemert, J.-M. Geusebroek, and A. W. Smeulders. The challenge problem for automated detection of 101 semantic concepts in multimedia. 2006.

[13] Y. Tagami. Annexml: Approximate nearest neighbor search for extreme multi-label classification. 2017.

[14] G. Tsoumakas, I. Katakis, and I. Vlahavas. Effective and efficient multilabel classification in domains with large number of labels. 2008.

[15] I. E. Yen, X. Huang, W. Dai, P. Ravikumar, I. Dhillon, and E. Xing. Ppdsparse: A parallel primal-dual sparse method for extreme classification. 2017.

[16] I. E.-H. Yen, X. Huang, P. Ravikumar, K. Zhong, and I. Dhillon. Pd-sparse : A primal and dual sparse approach to extreme multiclass and multilabel classification. 2016.