Extreme Multi-Label Classification for Text Classification

Annika Brundyn, Francesca Guiso, Tinatin Nikvashvili, Camille Taltas

May 19, 2020

1 Introduction

Extreme classification is a rapidly growing research area within machine learning focusing on multi-class and multi-label problems involving an extremely large number of labels. The goal of extreme multi-label classification is to learn a classifier which can assign a small subset of relevant labels to an instance from an extremely large set of target labels [12]. It should be emphasized that multi-label learning is distinct from multi-class classification, which only aims to predict a single mutually exclusive label. Multi-label learning allows for the co-existence of more than one label for a single data sample. In the extreme multi-label setting, it is significantly more challenging to simultaneously deal with extremely high input feature dimension and label dimension.

There are many real-world applications that require extreme classification. One application is document tagging in NLP. For instance, there are more than a million labels or categories on Wikipedia and each new article needs to be annotated with a subset of relevant categories. Other applications include video tagging and gene function prediction in bioinformatics [1].

In this project, we focus on an extreme classification task in the legal domain. The dataset is already pre-processed and the raw text is replaced by 5000 token ids and their corresponding TD-IDF weights. Each sample has a small subset of labels from 3993 possible labels.

There are two general approaches to multi-label learning: reduction-based and algorithm adaptation [6]. The reduction-based approach aims to reformulate the multi-label problem into multiple single label classification tasks. The benefit of this approach is that we are able to use simple binary classifiers. However, this approach treats labels as independent, which ignores label correlation thus often resulting in unsatisfactory performance. Additionally, as the number of classes grows to extremely large numbers, it becomes computationally challenging to fit a classifier for each class. Algorithm adaptation involves designing classifiers for the multi-label setting directly.

We tried both reduction-based and algorithm adaptation approaches to our problem, in addition to a simple baseline model, in which we used class frequencies to predict the labels. Our reductionist approaches include one-vs-all using linear SVM, and classifier chains using logistic regression. We then explored algorithms that inherently support multi-class, multi-label classification including random forest, k-nearest neighbors, ridge regression classifiers and neural networks.

In general we found the reduction-based approaches to be computationally infeasible, since we would have to fit nearly 4000 classifiers corresponding to each class. Additionally, each class is only observed in a handful of samples, which resulted in very weak binary classifiers. Of all the models evaluated, we found a simple multi-layer perceptron network to perform best on our dataset. Our final LRAP score achieved on the test set was 0.57.

2 Problem Definition

Data

Our data is derived from a multi-label text classification task in the legal domain. There are 5,000 features present in the dataset containing TF-IDF weights of words. In total, we have 3,993 classes, with an average of 5.3 labels per example.

Briefly, TF-IDF, short for "term frequency - inverse document frequency" is a numeric value intended to reflect how important a word is to a document in a corpus [11]. The TF-IDF value increases proportionally to the number of times a word appears in a document and is offset by the number of documents in the corpus that contain the word. This helps to adjust for the fact that some words appear more frequently in general and are not specifically relevant to the task.

Pre-processing

Since we did not perform the feature extraction from the raw data, we used the features as given, and did not use any additional feature preprocessing. A small number of examples (28 in the training set and 2 in the validation set) had corrupted string formatting for the labels. Since it was ambiguous how to process these examples, we removed them from both the training and validation sets.

We received the train, validation and test sets as three separate files and therefore did not implement any data splitting strategy of our own. The train and validation sets contain three columns, the first column contains example_id (corresponding to a given document), the second contains features that are comma separated and are in the form of 'token id: TF-IDF weight', and the last column contains comma separated labels associated with each example.

We converted these datasets into two sparse matrices, one containing features and another containing labels. The design matrix, X, contains 5,000 columns for each feature and each row represents an example. The y matrix contains 3993 columns for each label containing binary flags for positive class. If a word does not show up in a given document, the features column does not contain the TF-IDF for that word. Thus we chose to replace the missing values with zeros. The training dataset does not contain all 3993 labels, making prediction of those labels impossible for the validation set.

Evaluation Metric

We evaluate model performance using the label ranking average precision (LRAP). This metric calculates the average over each ground truth label assigned to each sample, of the ratio of true versus total labels with lower score [5]. This metric is useful for multi-label ranking problems, where the goal is to give better rank to the labels associated with each sample. The obtained score ranges between 0 and 1 with 1 corresponding to the best obtainable score.

3 Modelling and Experiments

In addition to a simple baseline model, our experiments explore two main approaches: (1) reductionist approaches, which reduce the multi-class multi-label problem to a binary problem, and (2) approaches that are inherently multi-class and multi-label. We implemented three different reductionist approaches: one-vs-all using linear SVM, classifier chain with logistic regression, and ensemble of classifier chain models. Our next set of experiments involved algorithms that inherently support multi-class and multi-label classification. We implemented and fine-tuned four different algorithms: random forests, k-nearest-neighbors, ridge regression, and multilayer perceptron.

Baseline Model

First, we defined a simple baseline model by assigning to each example a list of probabilities containing probabilities for each label, which were calculated by counting the number of positive cases for each label and dividing it by total number of examples.

One-vs-All

One-vs-all (OvA) is a general strategy that involves training a binary classifier for each of the classes present in the dataset and aggregating their predictions. In this project, we chose to fit Linear Support Vector Machines (SVM) as our binary classifier. The ith classifier finds the separating hyperplane between the ith class and all the other classes. A positive score is thus assigned to the predicted positive examples for class i and a negative score to all other examples. We implement the binary SVM classifiers using the liblinear library, which scales well with large datasets because it does not rely on kernelization. We use the SVM primal form (as we have more samples than features) with L2 penalty and the hinge loss function.

The key assumption in OvA is that each class is linearly separable from the rest and that there is no underlying correlation between classes. These are strong assumptions to make, especially when we do not know what the labels represent.

Classifier Chain with Logistic Regression

The naive assumption of label independence often does not hold in practice since some labels are far more likely to co-occur (See Figure 1). To account for correlated labels, we decided to implement a Classifier Chain (CC) model.

The CC model is still a reduction-based method, i.e. it fits a binary classifier for each class. Each binary classifier assigns a probability of membership to the specific label that presents the positive class. The binary classifiers are linked in a chain where the first classifier trains on all features available in the dataset and the next classifier trains on extended feature space that includes the predicted label from the previous classifier [9].

The advantage of the CC model is that we can use any algorithm to train each binary classifier in our chain while also accounting for label correlations by fitting the binary classifiers sequentially.

However, for both OvA and the classifier chain, computational complexity is a significant drawback. Since both models train a classifier for each label, this does not scale with a huge number of labels. Furthermore, since the classifiers are chained sequentially, training of the CC model cannot be parallelized either.

In this project we fit the CC model using Logistic Regression (LR) to predict binary relevance for each label. Since the order of the chain affects the accuracy of the CC model, we tried to visualize correlation between labels in the efforts to find the best ordering of labels. However, calculating cross-correlation for a total of 3,993 labels is computationally inefficient. Thus, we trained 4 CC models using a random ordering of classifiers in hope of potentially finding one good ordering. We used the NYU Prince Cluster to train the CC models in parallel, which took around 9 hours to complete. Therefore, using a more computationally intensive base classifier like Random Forest or SVM was not feasible. We also could not hyperparameter tune the LR model.

Random Forest Classifier

Random forest is an ensemble tree-based learning algorithm [4]. The model trains a large number of decision trees on bootstrap samples, where each bootstrap sample draws n data points randomly with replacement from the full training set of size n. Additionally, to maximize variance reduction by combining predictions of multiple decision trees, a random subset of all features are considered for splitting each node in each decision tree resulting in less correlated decision functions. The final predictions are made by averaging the predictions of each individual tree.

To control the complexity of each tree and avoid overfitting, we tuned two key hyperparameters: the maximum depth of the individual trees, and the number of features to use for splitting each node.

K-Nearest Neighbors Classifier

K-nearest neighbors (KNN) is a simple, non-parametric algorithm that stores all available cases and classifies new cases based on a similarity measure or a distance function

[2]. In the classification phase, k is an integer value specified by the user, and an unlabeled sample is classified by assigning the label which is most frequent among the k training samples nearest to that sample, as measured by the chosen distance function.

We tuned two key hyperparameters: the number of neighbors (k) and the weight function used in prediction. The optimal choice of k is highly data-dependent: in general a larger k suppresses the effects of noise, but makes the classification boundaries less distinct. The basic KNN uses uniform weights: that is, the value assigned to a data point is computed from a simple majority vote of the nearest neighbors. Under some circumstances, it is better to weight the neighbors such that nearer neighbors contribute more to the fit. We experimented with both a uniform weight function as well as assigning weights proportional to the inverse of the distance from the data point.

Ridge Regression Classifier

Ridge regression classifier adapts ridge regression to classification problems. Ridge regression is a linear regression model, which adds an L2-penalty to the model's weights. As in least squares linear regression, the ridge regression algorithm is a minimization problem: we minimize the mean squared error between the target and the predicted value plus the L2-norm of the coefficients. A regularization parameter, $\alpha \geq 0$ controls the weight attributed to penalizing the coefficients. To perform classification, the target values are converted to discrete values before fitting the regression model. In the multi-class, multi-label setting, the classifier fits a multivariate regression model, which outputs a vector of predictions for a given example.

We implemented the ridge regression classifier using sklearn's cross-validation to tune the regularization parameter on the training set. In addition to tuning the alpha, we also tuned the total number of folds used for cross-validation on the validation set.

Multi-layer Perceptron

Finally, we implemented a basic fully-connected multilayer perceptron neural network [10]. Our input layer consists of 5000 neurons, one for each feature, and the output layer contains 3993 neurons corresponding to each class label. The model supports multi-label classification, since for each class, the raw output passes through the logistic function as the final activation function (as opposed to a softmax function). The hidden layers use a ReLU activation function. The network trains using an adapted form of gradient descent and the gradients are calculated using backpropagation. We minimize the cross-entropy loss during training. For optimization, we used ADAM [8].

The parameters we tuned were the number of hidden layers, the number of hidden units in each hidden layer, as well as the initial learning rate used.

Other approaches considered

An alternative reduction-based approach to one-vs-all (OvA), is the all-vs-all (AvA) approach [2]. Given that our dataset contained 3993 classes, using an AvA approach would imply training $\binom{3933}{2}$ classifiers. Despite the fact that AvA uses fewer examples to train each classifier, it is considerably slower than OvA. Therefore, we decided to discard this approach.

We considered using Error-Correcting Output Codes (ECOC), an algorithm which encodes labels more efficiently that OvA. However, this method does not have an obvious multi-label implementation and, like OvA, it relies on independence of the predicted labels [7]. Given the poor performance achieved with OvA SVM, and with limited knowledge about the significance of labels, we did not want to implement another model which relies on this assumption.

4 Results and Discussion

Baseline

We evaluated our baseline model on the validation set and got a LRAP value of 0.059.

One-vs-all

The multilabel OvA SVM performed poorly, with an LRAP between our predicted labels and the ground truth of approximately 0.01. With limited computational speed, we were not able to fine-tune the implementation of this algorithm. The following sections discuss why we believe other methods are better-suited to this task.

Given the large number of classes in our dataset, implementing OvA is computationally very slow. Moreover, OvA works best with a small number of classes, and it is sensitive to class imbalance in the data, which can lead to bias in favor of the majority class. We chose not to pursue OvA classifier for this task, though we expect some improvements over the current implementation by incorporating class weights, which would make misclassification more costly for minority classes.

Classifier Chain

Each CC model gets a LRAP score of around 0.021-0.022 on the validation set. We also combined predictions of each model by averaging probabilities but this did not increase the LRAP score. Such a low LRAP score can be due to the fact that we could not hyperparameter tune the logistic regression model or find an ordering of labels that we could benefit from. Thus we conclude that due to computational complexity, the CC model is not the best approach for such a huge number of labels.

Random Forest

For random forest we compared twelve different combinations of hyperparameters. We tried restricting the maximum tree depth to the following values: $[30, 40, 50, 60, 70, 80]$. The number of features to consider at each node split were either $\sqrt{d}$ or $\log_2 d$, where d is the total number of features. Both of these values are considered good defaults in the literature. From Figure 2, we can see that when we increase the maximum tree depth to be greater than 50, the model's performance worsens significantly. Allowing the individual trees to become too complex encourages overfitting and leads the model to perform poorly on the validation set. We found $\sqrt{d}$ to perform better than $\log_2 d$. Empirically, this is a well known default [3].

Overall, the best LRAP score obtained was 0.398 which corresponds to setting the maximum tree depth to 40 and using the square root of the total number of features for each node split.

K-Nearest Neighbors

For the KNN model, we tried the following values for k, the number of neighbors: $[2, 5, 10, 20, 50, 100]$ as well as both uniform and distance-based weight functions.

From Figure 3, it is very clear that the distance-based weighting far outperforms the default uniform weighting. This makes sense in the context of a multi-label problem with correlated labels. For both weightings however, the optimal number of neighbors is 10. Using fewer than 5 or more than 30 neighbors drastically worsens our performance on the validation set which makes sense given the bias-variance tradeoff.

Our best result corresponds to an LRAP score of 0.471 on the validation set, using 10 neighbors and a distance-based weight function.

Ridge Regression

For the ridge regression model, we tuned the regularization parameter over the values 0.0001, 0.01, 0.1, 1.0, 2.0, 5.0, and 10.0 using the built-in cross validation function on the training dataset. Additionally, we tuned the number of folds for values 15 and d (Leave-One-Out) and tested their performance on the validation set. The best regularization parameter gave us a validation performance of 0.51 on 15 folds and 0.50 on d folds. These results are extremely close which makes sense given that we already reduced the variance on the validation set by using cross-validation on the regularization parameter.

Multilayer Perceptron

For the neural network, we tried using 1, 2, and 3 hidden layers with each of the layers containing either 100 or 500 hidden neurons. We also experimented with initializing the learning rate to either 0.001 and 0.0001 for the ADAM optimizer. Figure 4 below shows the validation set performance of the networks containing 2 and 3 hidden layers, each with either 100 or 500 neurons per hidden layer. It is important to note that these networks were trained for a very short amount of time, the maximum number of epochs have been limited to 200. Overall, we see a trend that increasing the size of the network - through the number of hidden layers and the number of hidden neurons per layer, improves the performance of the model. We tried training a network with a 1000 neurons per hidden layer but due to computational constraints, we were unable to train the model for long enough. Figure 4 also shows that the smaller learning rate achieved better performance.

Of the networks implemented, the best LRAP score of 0.579 was achieved using two hidden layers, each containing 500 neurons, and initializing the learning rate to be 0.0001.

5 Conclusion

This paper has focused on exploring methods for extreme multi-label classification tasks. The challenges of extending traditional classifiers to the multi-class, multi-label setting are drastically amplified when dealing with both extremely high input feature dimension and label dimension.

We consider two general approaches to multi-label learning. The first involves reducing the multi-class problem, into separate binary classifiers for each class, and aggregating the results from all the individual classifiers when predicting. The second approach is to use algorithms that are inherently designed for multi-class and multi-label problems.

We implemented three reduction-based approaches: one-vs-all using linear SVM, classifier chain using logistic regression and an ensemble of classifier chain models. We found all three of these techniques to be extremely computationally inefficient. Given that we had close to 4000 classes, fitting 4000 binary classifiers was slow even if each binary classifier trained relatively quickly. Additionally, neither SVMs or Classifier Chains are parallelizable. The models performed poorly based on the LRAP score. However, it should be considered that due to computational constraints these models were never fully trained, and no hyperparameter tuning was done. We concluded that given the large number of classes in our problem, these approaches were not feasible.

We then implemented algorithms that inherently support multi-class and multi-label predictions We implemented and fine-tuned four different algorithms: random forests, k-nearest-neighbors, ridge regression, and multilayer perceptron. These approaches were computationally faster, and achieved significantly better results on average.

Our best performing model was a simple fully-connected neural network, or multi-layer perceptron, with two hidden layers each containing 500 hidden units. We applied a ReLU activation function for the intermediate hidden layers and a logistic function on the final output layer in order to predict multiple labels for each sample.

Due to time and computational constraints, we were unable to try more complex network architectures, experiment with learning hyperparameters or train the network until convergence. Given more time, we would have liked to explore the neural network approach further. Deep learning solutions to extreme classification tasks is a very active area of research and there are a number of recent proposed architectures that we believe could have achieved better performance on this task [1, 12].

6 Appendix

Group Members and NetIDs

Group Name: CloveHouse

- Annika Brundyn (ab8690)
- Francesca Guiso (fg746) (responsible for submission)
- Tinatin Nikvashvili (tn709)
- Camille Taltas (ct2522)

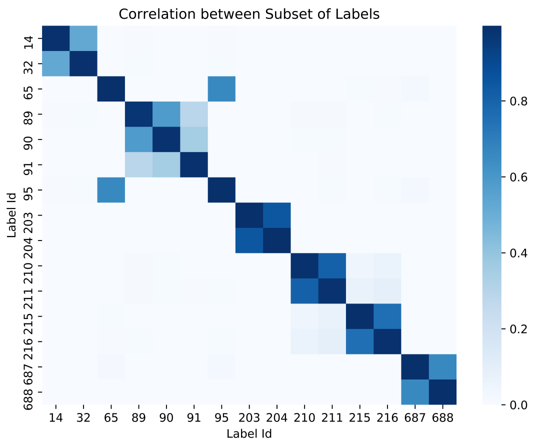

Figure 1: Correlation (using Cramér's V) Matrix of a Subset of Labels

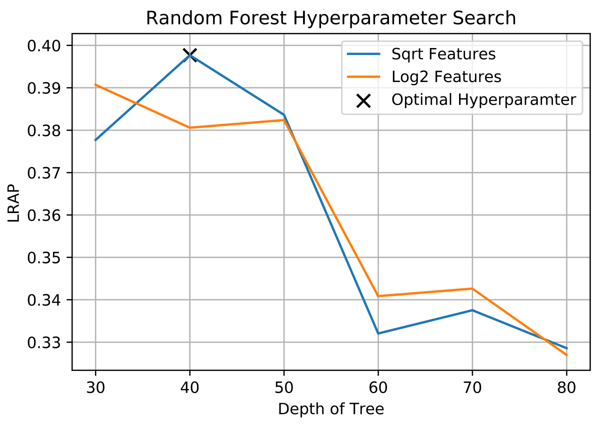

Figure 2: Random Forest Classifier

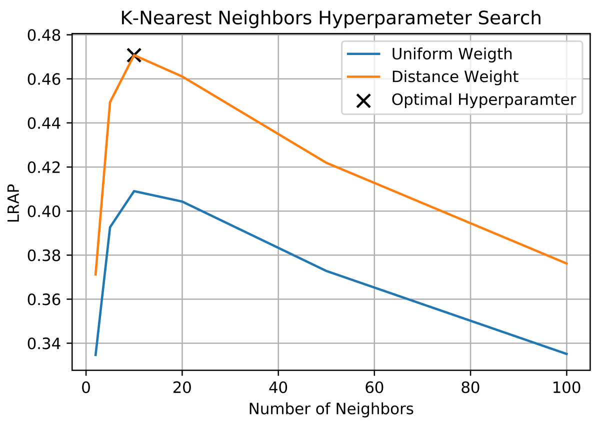

Figure 3: K-Nearest Neighbors

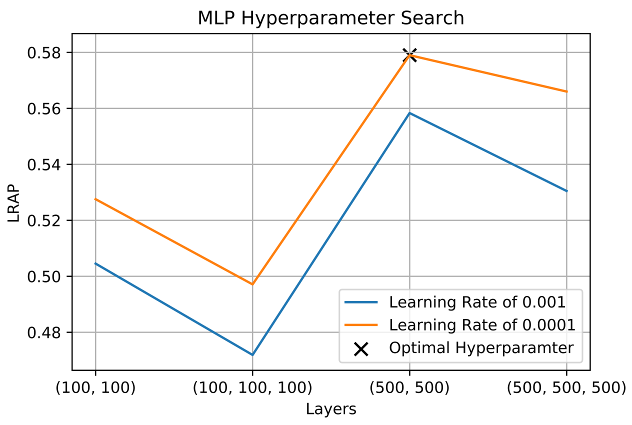

Figure 4: Multi-Layer Perceptron

References

[1] S. Bengio et al. "Extreme Classification". In: (2019).

[2] C. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006. ISBN: 0387310738.

[3] Leo Breiman. "Manual on setting up, using, and understanding random forests v3. 1." In: (2002). DOI: `https://www.stat.berkeley.edu/~breiman/Using_random_forests_V3.1.pdf`.

[4] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[5] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[6] Amit Daniely, Sivan Sabato, and Shai Shalev-Shwartz. "Multiclass Learning Approaches: A Theoretical Comparison with Implications". In: *CoRR* abs/1205.6432 (2012). arXiv: `1205.6432`. URL: `http://arxiv.org/abs/1205.6432`.

[7] Park SH. Fürnkranz J. "Error-Correcting Output Codes as a Transformation from Multi-Class to Multi-Label Prediction." In: (2012). DOI: `https://link.springer.com/chapter/10.1007/978-3-642-33492-4_21`.

[8] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: `1412.6980`.

[9] Jesse Read et al. "Classifier Chains for Multi-label Classification." In: (2009). DOI: `https://doi.org/10.1007/978-3-642-04174-7_17`.

[10] D. Rumelhart, G. Hinton, and R. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986). DOI: `https://doi.org/10.1038/323533a0`.

[11] Gerard Salton and Michael J McGill. "Introduction to modern information retrieval". In: (1986).

[12] Wenjie Zhang et al. "Deep Extreme Multi-label Learning". In: *CoRR* abs/1704.03718 (2017). eprint: `1704.03718`. URL: `http://arxiv.org/abs/1704.03718`.