Large Scale Multi-label Text Classification Problem

Yaqing Li
Department of Management Science and Engineering
Stanford University
94305, CA
yaqing2@stanford.edu

Ruoxi Zhang
CCRMA, Department of Music
Stanford University
94305, CA
ruoxi17@stanford.edu

Abstract

Large scale multi-label text classification refers to the problem of predicting multiple most related labels for a document, which remains challenging in that the dimension of the label space could reach millions. Recently, several solutions, including embedding-based, tree-based, and deep learning approaches, have been proposed to address this problem. This project implements the embedding-based method SLEEC and the deep learning method XML-CNN model purposed by previous papers. The result that the CNN models outperform others on precision of top-k predicted labels is in consistent with the previous papers models. This project further studies the general performance of the models and tests the possibilities of integrating embedding-based and deel learning methods taking the advantages of low-dimensional label space and using the context information.

1 Introduction

Text classification is one of the foundation problem in natural language understanding. Among many branches of text classification, large scale multi-label text classification or extreme multi-label text classification (XMTC) refers to the problem of assigning to each document its most relevant subset of class labels from an very large scale of label collection, where the number of labels could reach hundreds of thousands or millions. For example, there are millions of labels on Wikipedia web page and a large scale multi-label classifier can enable us to automatically categorize a new web page or article based on its text. Similar applications of large scale multi-label classification remains in recommendation systems. Multi-label classifiers can help us classify and label the reviews and rating text from customers.

Different from multi-class classification, multi-label classification assigns to each sample a set of target labels. This can be thought as predicting properties of a data-point that are not mutually exclusive, such as topics that are relevant for a document. A text might be about any of religion, politics, finance or education at the same time or none of these. Generally speaking, multi-label classification is more complicated than multi-class classification problem. The model structure, loss function, and optimization method for multi-lable problems should be designed to address the labels that are not mutually exclusive.

Large scale of labels makes the problem even harder. One of the main challenges in large scale multi-label classification is severe data sparsity. The number of the total labels cam achieve millions and billions, making it difficult to learn the dependency relation between text data and labels reliably. Another problem for large scale multi-label classification is the computational cost in both training and testing due to very sparse label matrix. Memory consumption during training is always a concern for large scale multi-label image classification.

In this project, we are aiming to address large scale multi-label text classification problem. Two kinds of state-of-the-art models: SLEEC and XML-CNN are implemented by ourselves to test their performances and analyse their strength and weakness. We also propose an integrated model which trying to combine the strength of both SLEEC and XML-CNN models. In the following part of the paper, we will introduce the related work, our proposed methods, and results and analysis.

2 Related Work

There are several categories of solutions to multi-label classification, including embedding based methods, tree-based ensemble methods (Prabhu and Varma, 2014), and deep learning approaches. We mainly focus on embedding based methods and deep learning methods.

The main idea of embedding based methods is to project the high-dimensional label vector to lower-dimensional, which is called compression, and the inverse process is called decompression. Some predecessor approaches of compression include compressing sensing (Hsu et al., 2009), label selection (Balasubramanian and Lebanon, 2012). Principal Label Space Transformation (PLST) (Tai and Lin, 2012) introduces a linear encoding and decoding based method to re-

duce the dimension of label space by finding key linear correlations between labels. PLST attempts to linearly maps label vectors to embedded label vectors by using the principal reduction of the original label matrix. Apart from encoding and decoding projections, another category of embedding methods is based on selecting important labels. An efficient embedding based approach is introduced to reduce the dimension of label vector by a randomized sampling procedure of important labels (Bi and Kwok, 2013). It is built on top of column subset selection problem (CSSP), which seeks to find a subset of columns among all columns of the original label matrix to span the original matrix as much as possible. The sparse local embeddings for extreme multi-label classification (SLEEC) (Bhatia et al., 2015) extends the embedding based method to address both prediction accuracy and scale problems in multi-label problem by extending the traditional embedding based methods. Since we implement this in out project, see section 4 for details.

The methods discussed above are all based on bag-of-word representations of text. The fact that bag-of-word representation disregard the grammar and word order is the fundamental limitation of those models. Recent progress in large scale multi-label classification is trying to overcome such a limitation by involving deep learning into the model. Deep learning models have achieved great results in lots of natural language understanding tasks, such as question answering, natural language inference, and machine translation. Recently, various kinds of deep learning structures have been applied in multi-class text classification problems and those models have outperformed than linear classifiers with word representations. The most commonly used deep learning models are recurrent neural network (RNN), convolutional neural network (CNN), and the combination of these two models. Although some of the deep learning models for multi-class classification are also tested on multi-label dataset, they didn't included the multi-class settings in optimization and thus didn't achieve good results. Despite the large amount of applications in multi-class problems, deep learning models aiming to solve multi-label classification is still at the beginning stage. Therefore, we will first talk about some multi-class models and then multi-label models.

CNN-Kim (Kim, 2014) was applied to sentence classification, which improved the state-of-the-art performance on 4 out of 7 tasks. The basic structure of the model is a convolutional layer followed by a max pooling layer, and then followed by a fully-connected softmax layer that ouputs the labels. And the author experimented with 4 variants of the model: CNN-rand, CNN-static, CNN-non-static, and CNN-multichannel. From CNN-rand to CNN-multichannel, more flexibility is added. CNN-rand is the baseline model where all of the weights are randomly initialized. CNN-static used pre-trained word2vec embedding, and all of the words remain the same during the training. CNN-non-static also used pre-trained word2vec embedding, but the word vector is trained individually for each task. CNN-mulichannel is the most complex model which trained two sets of word vectors that serves as two channels of the model.

Although the model structure is very simple, it achieved excellent performances in several datasets including MR (movie reviews), SST-1 and SST-2 (Stanford Sentiment Treebank), and has 1% to 5% improvements than several RNN models and tree-based models. Among the 4 variants of the CNN model, CNN-static, CNN-non-static and CNN-multi channel have similar performances, and all of the three outperforms the baseline CNN-rand model.

XML-CNN (Liu et al., 2017) was proposed to address the multi-label text classification problem. XML-CNN, based on CNN-Kim, is the first attempt that utilizing CNN models for multi-label classification problems. Since we implemented a modified version this model in our project, details of the XML-CNN model structure are discussed in Method Section. The deep learning approaches differ a lot from the traditional embedding-based and tree-based models since it use the raw context information of texts instead of the bag-of-word representation. The key idea of deep learning models for text classification is that it extracts and utilizes more word sequence and grammar information for the raw text, which will then have better performances than traditional embedding-based and tree-based methods.

Based on the literature review,we had the following conclusions of the current research:

- Current methods to address the large scale multi-label classification problem can be categorized into embedding-based and deep learning methods.
- Embedding-based method reduce the dimension of the labels, which solve the data sparsity problem.
- Deep learning models to address this problem are mainly CNN models. Those CNN models have relatively simple structures, usually one convolutional layer.

3 Data

3.1 Datasets

We decides to use three common textual datasets and they are shown in Table 1. They can be downloaded from the extreme classification repository [1]. Wiki10 and AmazonCat are the most commonly used datasets for large scale multi-label classification, both of them have labels in 10k level and includes raw text and bag-of-word information.

[1] http://manikvarma.org/downloads/XC/XMLRepository.html

Dataset	#instances	#features	#labels
Bibtex	7,395	1,836	159
Wiki10-31k	20,762	101,938	30,938
AmazonCat-13k	1,493,021	203,882	13,330

Table 1: Common datasets in multi-label classification

The Bibtex dataset, with only 159 labels, is actually not a large scale multi-label text dataset, and it only contains only bag-of-word information. We implemented SLEEC model on Bibtex to deeply understand the performance of SLEEC.

3.2 Data pre-processing

For Bibtex dataset, we split the data into train, dev, and test sets with different ratios since Bibtex is small and the others are large.

The AmazonCat-13K dataset is a huge text dataset with 13k labels and 1M training points. To apply the deep learning model, we extract the raw review text from the dataset for each product and mapping the review text with the labels. Then we split the data to train, dev, and test datasets. Since the labels are really sparse, we stored the labels with sparse matrices.

The raw data in Wiki10-30K dataset is mixed with HTML tags and Wiki page information. Thus we firstly write a content extractor to filter away the messy unrelated information and then index the word for later use.

For all datasets, we choose the train:dev:test ratio as 6:2:2.

4 Method

The primary goal for this project is to implement the embedding-based and deep learning model proposed by previous researchers to test their performances on various datasets. In this project, the SLEEC and XML-CNN approaches are re-implemented with more careful evaluations on the models.

More importantly, the second goal of this project is to combine embedding-based and deep learning models. One of the drawbacks of deep learning models is that they still need to compute and store the large multi-hot label matrix. Embedding-based model serves as a good approach to reduce data sparsity since it reduce the dimension of the labels. Therefore, we propose to use the reduced embedded labels output by SLEEC as the target of CNN. We hope it can increase the CNN training speed without large compromise of the accuracy. Since SLEEC embedding needs huge memory, we first cluster the samples and then run cnns on each cluster.

4.1 Models

We proposed to implement the following models:

- Baseline Model: random forest classifier.
- Embedding-based Model: SLEEC model (Bhatia et al., 2015).
- Deep learning Model: XML-CNN model (Liu et al., 2017). Different from the original XML-CNN model, we made our own modifications. Firstly, instead of training the wording embedding from the raw text, we integrated per-trained glove vectors to increase the stability of XML-CNN model. Secondly, we increase the filters of the conv layer to add more complexity of XML-CNN model.
- Integrated Model: combining SLEEC and XML-CNN model. Use the reduced embedded labels output by SLEEC as the target of XML-CNN.

The detailed model structures are described in the following subsections.

4.1.1 Baseline model

Random forest classifier is chosen for the baseline model since it supports multi-label problems, overcomes the overfitting problem from decision tree, and it is simple and efficient. The results of the baseline model serve as references to evaluate our proposed models.

4.1.2 SLEEC

The sparse local embeddings for extreme multi-label classification (SLEEC) (Bhatia et al., 2015) is introduced to address both prediction accuracy and scale problems in multi-label problem by extending the traditional embedding based methods. For a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i), \cdots (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$, it maps the label vectors $\boldsymbol{Y} = [\boldsymbol{y}_i \cdots \boldsymbol{y}_i] \in \mathbb{R}^{L \times n}$ to embeddings $\boldsymbol{Z} = [\boldsymbol{z}_i \cdots \boldsymbol{z}_i] \in \mathbb{R}^{\hat{L} \times n}$. We re-implement the model by ourselves based on the paper to examine the ability of this model since like most of the methods in this area, precision at k (P@k) is used as the only evaluation metric, which is not sufficient to evaluate the performance of a model.

Learning embeddings

Instead of linearly projecting or label-selecting the original label vectors (denoted as $\boldsymbol{y}_i \in \mathbb{R}^L$) onto a lower-dimensional subspace, it attempts to learn the embeddings $\boldsymbol{z}_i \in \mathbb{R}^{\hat{L}}$ of the label vectors by preserving the pairwise distance between k-nearest neighbors, i.e., the object of the learning process is to preserve $\mathrm{d}(\boldsymbol{y}_i, \boldsymbol{y}_j) = \mathrm{d}(\boldsymbol{z}_i, \boldsymbol{z}_j), \forall i$, j representing the index of ks closest neighbors of the ith data sample, and $\mathrm{d}(\cdot)$ representing the distance metric (cosine distance). Regressors $\boldsymbol{V} \in \mathbb{R}^{\hat{L} \times d}$ are learned by $\boldsymbol{V}\boldsymbol{x}_i = \boldsymbol{z}_i, \forall i$. The objective function with l_1 regularization of $\boldsymbol{Z}$ is:

$$\min_{\boldsymbol{Z}} \|P_\Omega(\boldsymbol{Y}^T\boldsymbol{Y}) - P_\Omega(\boldsymbol{Z}^T\boldsymbol{Z})\|_F^2 + \lambda\|\boldsymbol{Z}\|_1 \quad (1)$$

where Ω is the set of neighbors to preserve, indicating that $(i,j) \in \Omega \iff j \in \mathcal{N}_i$, where $\mathcal{N}_i = \arg\max_{S,\ S \le \alpha \cdot n} \sum_{j \in S} (\boldsymbol{y}_i^T, \boldsymbol{y}_j)$; and operator $P_\Omega(\cdot)$ is

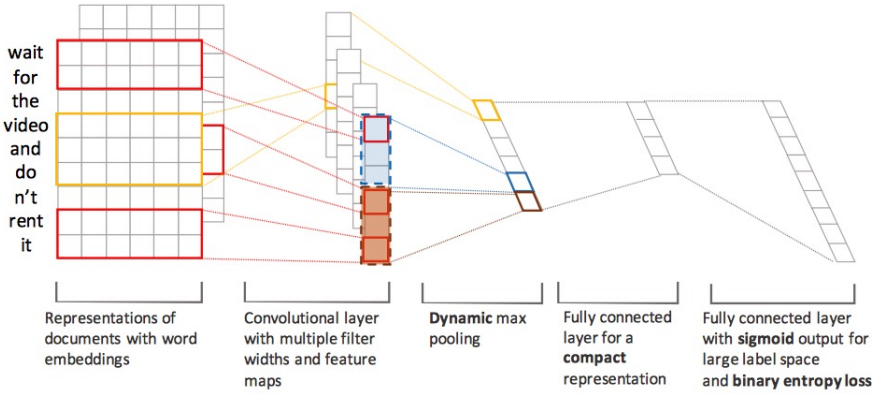


Figure 1: Model architecture of XML-CNN with an example sentence

a filter that filters out the faraway neighbors:

$$P_\Omega(\boldsymbol{Y}^T\boldsymbol{Y})_{i,j} = \begin{cases} \langle \boldsymbol{y}_i, \boldsymbol{y}_j \rangle, & \text{if } (i,j) \in \Omega \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

Since $\boldsymbol{V}$ is used to predict $\boldsymbol{Z}$ from $\boldsymbol{X}$, the objective can be represented as with l_2 regularization of V:

$$\min_{\boldsymbol{V}} \|P_\Omega(\boldsymbol{Y}^T\boldsymbol{Y}) - P_\Omega(\boldsymbol{X}^T\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{X})\|_F^2 + \lambda\|\boldsymbol{V}\|_f^2 + \mu\|\boldsymbol{V}\boldsymbol{X}\|_1 \quad (3)$$

Optimization

Considering that the objective function is non-convex and the scalability, the optimization is separated into two steps - the embeddings $\boldsymbol{Z}$ is first learned and then regressors $\boldsymbol{V}$ is learned given $\boldsymbol{Z}$, i.e., the objective for first step is:

$$\min_{\boldsymbol{Z}} \|P_\Omega(\boldsymbol{Y}^T\boldsymbol{Y}) - P_\Omega(\boldsymbol{Z}^T\boldsymbol{Z})\|_F^2 \quad (4)$$

and the second step is:

$$\min_{\boldsymbol{V}} \|\boldsymbol{Z} - \boldsymbol{V}\boldsymbol{X}\|_F^2 + \lambda\|\boldsymbol{V}\|_f^2 + \mu\|\boldsymbol{V}\boldsymbol{X}\|_1 \quad (5)$$

Equation 4 is optimized using singular value projection (SVP) (Jain et al., 2010), which is a projected gradient descent method to reduce the dimension of matrices. The idea of SVP is to reconstruct the matrix by its top eigenvalue decomposition, the gradient is determined by the difference of the original matrix and the reconstructed one. The update of SVP at step t is:

$$(\boldsymbol{Z}^T\boldsymbol{Z})_{t+1} = \boldsymbol{M}_{t+1} = P_{\hat{L}}\left(\boldsymbol{M}_t + \eta P_\Omega\left(\boldsymbol{Y}^T\boldsymbol{Y} - \boldsymbol{M}_t\right)\right) \quad (6)$$

where operator $P_{\hat{L}}(\cdot)$ means to take the top $\hat{L}$ eigenvalue decomposition.

Equation 5 is optimized using alternating direction method of multipliers (ADMM) (Sprechmann et al., 2013), which is a convex optimization algorithm that changes the problem of optimizing $\boldsymbol{V}$ into optimizing $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, i.e., iterating three operations - minimizing the augmented Lagrangian over $\boldsymbol{V}$ with $\boldsymbol{\alpha}$ and the Lagrange multipliers $\boldsymbol{\beta}$ fixed, minimizing the augmented Lagrangian over $\boldsymbol{\alpha}$ with $\boldsymbol{V}$ with $\boldsymbol{\beta}$ fixed, and update the Lagrange multipliers using gradient ascent.

The detailed optimization algorithm implementation can be seen on our GitHub [2].

Prediction

The prediction process uses kNN classifier in the embedding space. Given an unseen sample $\boldsymbol{x}^*$, the predicted embedding vector is $\boldsymbol{z}^* = \boldsymbol{V}\boldsymbol{x}^*$. Then the label vector of $\boldsymbol{x}^*$ is the sum the label vectors of the k nearest neighbors among all training samples, i.e., $\boldsymbol{y}^* = \sum_{i:\boldsymbol{V}\boldsymbol{x}_i \in \text{kNN}(\boldsymbol{V}\boldsymbol{x}^*)} \boldsymbol{y}_i$.

Scalability

Since the speed of kNN search is slow, SLEEC divides the training data into multiple clusters of size $\lfloor \frac{N_{\text{train}}}{6000} \rfloor$, learns the embeddings in each cluster separately, and kNN search in the prediction process is performed only in the cluster the unseen sample belongs to. For large scale dataset, the data samples are divided into smaller clusters considering the efficiency of the model. To avoid memory errors at ADMM, we add pincipal component analysis (PCA) to reduce the dimension of features. The SLEEC method also attempts to empirically compensate for the problem of unstable clustering in large dimensions by learning a small ensemble, where each individual learner is generated by a different random clustering.

4.1.3 XML-CNN

XML-CNN (Liu et al., 2017) is proposed to address the multi-label text classification problem. It first trains an embedding layer and then uses multiple filters with different sizes (filter size = 3, 4, 5) to build the convolutional layers. Filters only move along the word sequence to get the meaning of the word sentence, of

[2] https://github.com/ruoxi17/cs224u_fp/blob/master/sleec.py

which the rationale is similar to the n-bag of words. After the convolutional layer, a max pooling layer is followed and then a fully-connected layer is at the end.

The XML-CNN model is based on CNN-Kim model, and it has the following main components.

- Dynamic max pooling: The max pooling layer of the previous CNN models are usually pre-defined. The idea of the max pooling layer is capture the most important feature in a small part of the figure (eg. 4v pixels), so usually the maximum value in the area is picked as the results of the pooling layer. In XML-CNN model, instead of generating only one feature in the filter area, the pooling layer dynamically generates p features of the filter area, which maximize the useful information extracted. The parameter p is learned during the training process.

- Bottleneck layer: The bottleneck layer is used the reduce the number of parameters and memory consumed to train the model. In CNN-Kim model, pooling layer and output layer are directly connected. However in XML-CNN model, the authors propose to add a fully-connected layer with h units between pooling and output layer, referred to as the bottleneck layer. Since the number of hidden states in much less the size of pooling layer and number of labels in the output layer, this bottle neck layer significantly shrinks the number of parameters in the model.

- Loss function: Since the XML-CNN model is aiming to solve multi-label problems, it uses the binary cross-entropy over sigmoid instead of the cross-entropy model over softmax in multi-class classification models.

$$\begin{aligned} \text{Loss} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{L}[y_{ij}\log\left(\sigma(f_{ij})\right) + \\ (1-y_{ij})\log\left(1-\sigma(f_{ij})\right)] \\ \sigma(x) = \frac{1}{1+e^{-x}} \end{aligned} \tag{7}$$

The basic structure of XML-CNN model is described in Figure 1.

In our project, we modified the original XML-CNN model by using pre-trained word2vec (glove) to instead of training word embedding by ourselves to increase the stability of the model. Furthermore, we also test the performance of more complicated XML-CNN model structures with more filters in the convolutional layer. Specifically, we implemented three XML-CNN models:

- XML-CNN model 1: use pre-trained word embedding (glove50)

- XML-CNN model 2: increase embedding dimension to glove300

- XML-CNN model 3: increase # of filters from 3 to 6 with various filter sizes: 2, 3, 4, 5, 6, 7.

The detailed implementation of XML-CNN models and the following integrated model can be seen in Stanford box [3].

4.1.4 Integrated model

In this step, two types of models discussed above are combined by using the embedded label vectors as CNN output, taking advantages of using context information and low-dimensional output layer. The proposed model structure is as Figure 2.

This integrated model used the embedded labels learned with SLEEC. After implementing the label transform, the label is not sparse anymore and we can use mean square error as the loss function for CNN model. This combination is proposed to increase the speed and memory performance of CNN models without large compromise of the accuracy.

In the integrated model, the output of XML-CNN model is the embedded label vectors output from SLEEC. During the training proccess, only loss is computed. In evalution, we first utilize the XML-CNN to predict the embedded label vectors and then transform the vectors to real labels.

4.2 Evaluation metrics

Precision at k (P@k): P@k is a widely used metric used large scale in multi-label classification problem. It counts the fraction of correct predictions in the top k scoring labels in the predicted label vector.

Precision, recall, and f_1-score: In binary classification, precision is the fraction of relevant instances among the retrieved instances, recall is the fraction of relevant instances that have been retrieved over the total amount of relevant instances, and f_1-score considers both precision and recall, measuring the accuracy of the model. Previous papers rarely use these metrics, while we use them to test the general performance of the models.

5 Results

5.1 Results of BibTex dataset

Firstly, we run our baseline and SLEEC models on BibTex dataset, which uses bag-of-word information and is of small size. The results can be seen in Table 2. In general, our implementation of SLEEC model outperforms the baseline model and the results are slightly lower than the original paper. It should be mentioned that we tuned the model based on dev set and evaluate the model based on test set, while the original SLEEC paper has only train set and test set, and it did not say how the model is tuned. Our dev set results are slightly lower than the original SLEEC paper's results (this may

[3] https://stanford.box.com/s/f9yrcvqzbkrfuc9h48jxhakjbmm54e5t

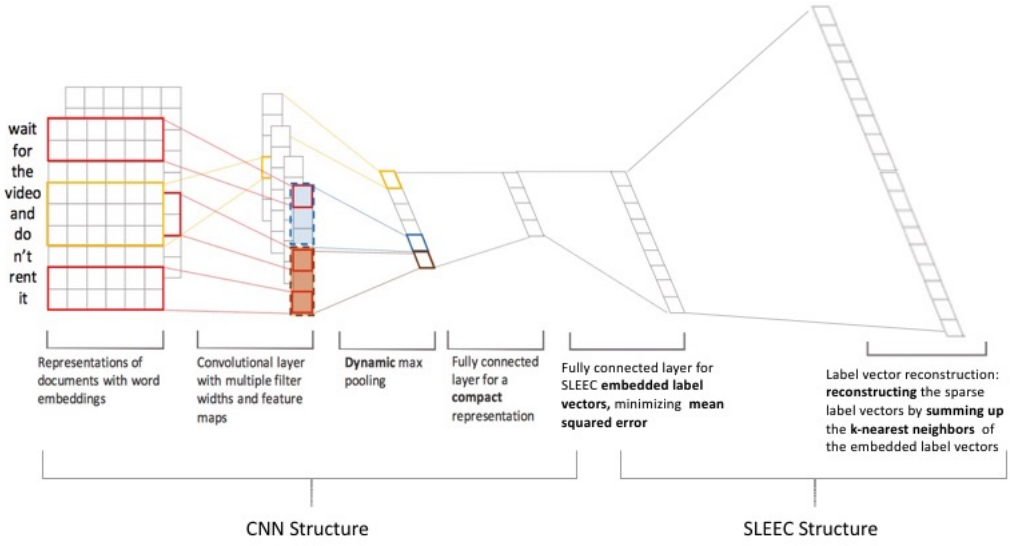


Figure 2: Proposed model structure for XML-CNN and SLEEC integrated model

result from the limited attempts of α at the preserving the nearest neighbors step since the paper does not give a reference); while our test set results are comparable with the dev set ones, indicating that the model is not overfitted.

Metrics	Baseline	$\text{SLEEC}_{\text{our}_{\text{dev}}}$	$\text{SLEEC}_{\text{our}_{\text{test}}}$	$\text{SLEEC}_{\text{org}}$
P@1	40.33	64.98	64.23	65.57
P@3	24.30	38.95	36.83	40.02
P@5	16.21	27.60	25.96	29.30
$\mathbf{F_1}$	0.343	0.418	0.404	-

Table 2: Results of Bibtex dataset

5.2 Results of Wiki10 dataset

	Baseline	SLEEC	CNN	CNN-glove300	CNN-multilayer	combined
P@1	73.19	77.29	81.18	**81.39**	81.34	79.57
P@3	40.28	59.08	60.62	60.64	**61.85**	41.38
P@5	25.40	49.37	49.59	50.76	**51.40**	31.56
Precision	0.1543	0.1447	0.1794	**0.2189**	0.1762	0.1001
Recall	0.0678	**0.4664**	0.1053	0.1093	0.1272	0.2786
$\mathbf{F_1}$	0.0763	**0.1965**	0.1140	0.1109	0.1336	0.1473
Memory	$>$ 3G	$>$ 20GB	**1GB**	**1GB**	**1GB**	SLEEC part: $>$ 20GB CNN part: 100MB

Table 3: Results of Wiki10 dataset

Table 3 shows the results of our models using Wiki10 dataset. It should be mentioned that apart from CNN model and its variations, other model runs only on a subset of clusters (4/5), which spans about 60% of the original data. This is a compromise since we met memory errors when running on one large cluster on a 16 vCPUs, 60 GB memory Google cloud machine.

It can be learned that CNN models outperform SLEEC model at P@ks, while SLEEC has a larger F_1 score. The combined model has a comparable P@1, while precision drops when k is larger.

5.3 Results of AmazonCat dataset

We only run the CNN models on the AmazonCat dataset because the pre-clustering step in SLEEC need to operate all data entries, which causes unavoidable memory errors.

The results of XML-CNN on AmazonCat dataset is as follows. Different from the results in previous research, our P@ks are relatively low on AmazonCat dataset. This is because the actual number of labels after parsing the raw text reaches 100k, which is 10 times as large as what was mentioned in other papers. Making accurate prediction with larger scale of labels is far more difficult.

Metrics	XML-CNN
P@1	23.57
P@3	18.30
P@5	15.21

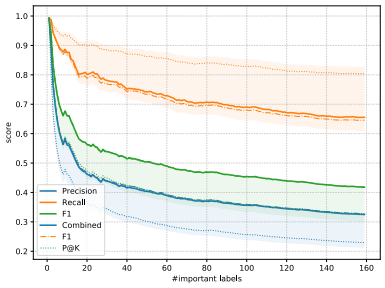
Table 4: Results of AmazonCat dataset

6 Analysis

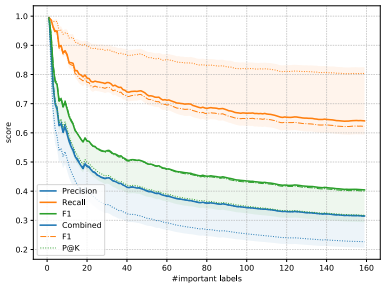
6.1 Evaluation metrics analysis - conflicting relationship between P@k and F_1 score

When tuning the SLEEC model, we find a conflicting relationship between P@k, the ability of choosing top labels, and F_1 score, which shows the general performance of the model, i.e., the best P@k values may not indicate a good overall performance. Figure 3 shows the precision, recall, and f_1 curves over the number of important labels that taken into consideration when calculating the scores.

The original SLEEC paper measures only the P@k values, whereas we measure the F_1-score to show the general accuracy of the model. As seen in Figure 3, the model that finds the maximal P@k may not be the optimal one. We tune our model with an objective to finds large P@k as well as large F_1-score. As shown in Table 3, although P@k values may not evaluate the general performance of the model, F_1-score might be



(a) dev set



(b) test set

Figure 3: The precision, recall, and F_1-scores for Bibtex, line styles indicating sorting criteria to find the best parameter combinations: solid - based on the combination of F_1-score and P@1; dash dotted - based on f_1-score only; dotted: based on P@1, P@3, and P@5 (P@k). The solid lines are the results for the final tuned model.

too strict for evaluating the large scale multi-label classification problem since choosing all (recall) the correct (precision) labels from 30K candidates is challenging.

6.2 Overall results analysis

As shown in Table 3, CNN models outperform others in P@ks. However, although the precision is high, the F_1 score is much lower than the SLEEC model because the recall is low. In contrast, the SLEEC model reach a higher F_1 score because the recall is much higher. This indicates that the behaviours of two types of models are different - the SLEEK model may introduce more negative positives by the kNN label reconstruction methods (which sums up the label vectors of the predicted embedding's k nearest neighbors, k tuned to 10 or 20 for different clusters) to reach higher P@ks; while CNN models attempt to predict more accurately at the cost of missing labels, i.e., more false negatives.

When comparing memory usage, the CNN models outperform others a lot. This may result from our implementation of the SLEEC. The SVP optimization step requires the memory of $O(n^2)$, where n is the number of samples in a cluster, and the ADMM step requires the memory of $O(d^2)$, where d is the feature dimension, which could be as large as 70K for a Wiki10

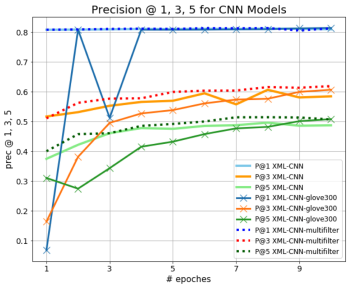


Figure 4: P@ks for XML-CNN models

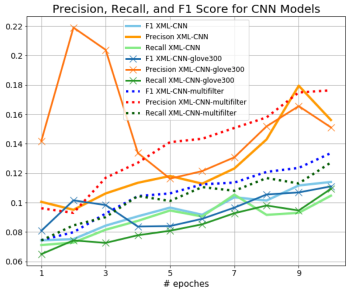


Figure 5: Precision, Recall and F1 score for XML-CNN models

cluster. We use PCA to reduce the dimension of the features, making the approach more scalable at the cost of performance drop about 5% comparing to the original paper. The original paper implementation of SLEEC of ADMM uses an external library, which we could not take for reference.

The integrated model inherits the features of high P@1 from CNN and lower precision, higher recall from the SLEEC label reconstruction methods. In general, the performance of this method is not as competitive as SLEEK or CNN model, but it is much more scalable in terms of memory usage and running speed (if not considering the embedding step). It should be clarified that the embedding method may not limited to SLEEC, i.e., the goal of this integrated model is to find the possibility of making deep learning method more scalable using label embedding. We choose SLEEC because of the time limitation, and more embedding methods with good memory or time performance could be tested.

6.3 Analysis of XML-CNN models

As shown in Figure 4, high P@1 is achieved quickly while P@3 and P@5 are improved gradually during training for all of the CNN models. It can also be found that the CNN model with multiple filters, which has the more complicated convolutional structures, achieves more stable P@ks than other CNN models. This performance make sense since adding more filters means the model is utilizing more word combination and sequence information. In addition, increasing the dimension of word embeddings did not improve the performance of CNN models.

As shown in Figure 5, CNN models has very low F_1 scores and recalls. Results showed that it is hard for CNN to catch all of the labels, which might be caused by the high label density in the Wiki10 datasets. Also, similar to P@ks, model with more filters in convolutional layers has the best performance of precision, recall and F_1.

7 Conclusion

Based on the above results and analysis, we have the following conclusions.

XML-CNN models outperform all of the other models on P@ks. However, despite the high P@ks, F_1 and recalls for CNN models are very low. We also found that adding more filters in XML-CNN model can increase the stability of the model.

Combined CNN and SLEEC model compromise the accuracy with a gain of memory and time performance. More embedding models could be tested in the future to replace SLEEC to find a better way to increase the scalability of deep learning based methods. For the evaluations, we find that precision@k may not be sufficient to indicate the performance of the model, while traditional F_1-score might be too strict for this problem. In the future, more representative evaluation metrics could be examined to give a clearer view of the performance of a model. Additionally, from the finding of the low-precision high-recall feature resulting from the SLEEC label reconstruction method, more studies could be conducted on the direction of mapping label embeddings to multi-hot label vectors, this may well increase the performance of general embedding based methods, and add possibilities to develop new scalable deep learning based methods.

References

Krishnakumar Balasubramanian and Guy Lebanon. 2012. The landmark selection method for multiple output prediction. *arXiv preprint arXiv:1206.6479*.

Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. 2015. Sparse local embeddings for extreme multi-label classification. In *Advances in Neural Information Processing Systems*, pages 730–738.

Wei Bi and James Kwok. 2013. Efficient multi-label classification with many labels. In *International Conference on Machine Learning*, pages 405–413.

Daniel J Hsu, Sham M Kakade, John Langford, and Tong Zhang. 2009. Multi-label prediction via compressed sensing. In *Advances in neural information processing systems*, pages 772–780.

Prateek Jain, Raghu Meka, and Inderjit S Dhillon. 2010. Guaranteed rank minimization via singular value projection. In *Advances in Neural Information Processing Systems*, pages 937–945.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. 2017. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–124. ACM.

Yashoteja Prabhu and Manik Varma. 2014. 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*, pages 263–272. ACM.

Pablo Sprechmann, Roee Litman, Tal Ben Yakar, Alexander M Bronstein, and Guillermo Sapiro. 2013. Supervised sparse analysis and synthesis operators. In *Advances in Neural Information Processing Systems*, pages 908–916.

Farbound Tai and Hsuan-Tien Lin. 2012. Multilabel classification with principal label space transformation. *Neural Computation*, 24(9):2508–2542.