LibMultiLabel User Guide

LibMultiLabel Project Authors*

January 10, 2024

1 Introduction

LibMultiLabel is a library for multi-label text classification. It provides end-to-end services consisting of data preprocessing, model training, and parameter selection.

Documents of LibMultiLabel including command line and API usage are available at

`https://www.csie.ntu.edu.tw/~cjlin/libmultilabel`.

In this guide, through examples we give some practical tips in handling multi-label text classification.

Details of data sets used in this guide are in Table 1. They can be download from the LIBSVM data sets [1].

2 Parameter Selection for Neural Networks

The performance of a model depends on the choice of hyper-parameters. The following example demonstrates how the BiGRU model performs differently on the EUR-Lex data set with two parameter sets.

First, train a BiGRU model with the default configuration file example_config/EUR-Lex/bigru_lwan.yml with a little modification on the learning rate. Some important parameters are listed as follows.

```
learning_rate: 0.001
network_config:
  embed_dropout: 0.4
  encoder_dropout: 0.4
  rnn_dim: 512
  rnn_layers: 1
```

*See contributors at https://github.com/ASUS-AICS/LibMultiLabel/graphs/contributors

[1] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel.html`

Table 1: Data statistics

Data set	# training	# test	# labels
RCV1	23,149	781,265	101
EUR-Lex	15,449	3,865	3,956

The training command is:

```
python3 main.py --config example_config/EUR-Lex/bigru_lwan.yml
```

After training for 50 epochs, the checkpoint with the best validation performance is stored for testing. The average P@1 score on the test data set is 81.40%.

Next, the learning rate is changed to 0.003 while other parameters are kept the same.

```
learning_rate: 0.003
network_config:
  embed_dropout: 0.4
  encoder_dropout: 0.4
  rnn_dim: 512
  rnn_layers: 1
```

By the same training command, the P@1 score of the second parameter set is about 78.14%, which is 4% lower than the first one. This demonstrates the importance of parameter selection. For more striking examples on the importance of parameter selection; see Liu et al. (2021).

However, a complete search of the parameter space is often time-consuming, especially on A complete search of the parameter space is often time-consuming, especially on a large data set or a complex neural network. For example, consider the command for parameter selection.

```
python3 search_params.py
              --config example_config/EUR-Lex/bigru_lwan_tune.yml
```

In the configuration file, we specify a grid search on the following parameters to get the best BiGRU model on the EUR-Lex data set. We set the embed_cache_dir to .vector_cache to avoid downloading pre-trained embeddings repeatedly for each configuration.

```
learning_rate: ['grid_search', [0.003, 0.001, 0.0003]]
network_config:
  embed_dropout: ['grid_search', [0, 0.2, 0.4, 0.6, 0.8]]
  encoder_dropout: ['grid_search', [0, 0.2, 0.4]]
  rnn_dim: ['grid_search', [256, 512, 1024]]
  rnn_layers: 1
embed_cache_dir: .vector_cache
```

The process takes about 1 day on four Nvidia Tesla V100 GPUs to find the best parameter set of learning_rate=0.0003, embed_dropout=0.4, encoder_dropout=0.4, and rnn_dim=512. Details of other parameters are in the configuration file example_config/EUR-Lex/bigru_lwan.yml. Additionally, after the search process, the program applies the best parameters to obtain the final model by adding the validation set for training. The average P@1 score is 83.65% on the test set.

It is time consuming to search over the entire parameter space. To save time, LibMultiLabel has incorporated some early stopping techniques implemented in Ray (Liaw et al., 2018), which is a framework for parameter selection. Here we demonstrate an example of applying an ASHA (Asynchronous Successive Halving Algorithm) Scheduler (Li et al., 2020). First, uncomment the following lines in the configuration file example_config/EUR-Lex/bigru_lwan_tune.yml.

```
scheduler:
  time_attr: training_iteration
  max_t: 50
```

```
grace_period: 10
reduction_factor: 3
brackets: 1
```

Under the same computing environment and the same command, the best parameter set of learning_rate=0.001, embed_dropout=0.4, encoder_dropout=0.2, and rnn_dim=512 is found in 47% of the time compared to the grid search, while the average test P@1 score = 82.90% is similar to the result without early stopping. For more complete results of the above examples, please refer to Table 3.

3 Linear Classifiers are Competitive in Some Cases

While non-linear classifiers such as neural networks are now widely used for multi-label classification, we show that linear classifiers give competitive performance in some cases. We demonstrate that they are easy to use and require less training time.

3.1 Linear Classifiers in LibMultiLabel

Consider a set of training instances $\{(\boldsymbol{y}_i, \boldsymbol{x}_i)\}_{i=1}^{l}$ where l is the number of instances, L is the number of labels, n is the number of features, $\boldsymbol{x}_i \in \mathbb{R}^n$ is a feature vector, and $\boldsymbol{y}_i \in \{-1, 1\}^L$ is a label vector such that

$$y_{ij} = \begin{cases} 1, & \text{if } \boldsymbol{x}_i \text{ is associated with the label } j, \\ -1, & \text{otherwise.} \end{cases}$$

In LibMultiLabel, currently all techniques based on linear classification aim to learn a $f : \mathbb{R}^n \to \mathbb{R}^L$ which is composed of L decision functions.

$$f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_L(\boldsymbol{x})).$$

3.1.1 One-vs-rest (Binary Relevance)

The one-versus-rest setting, also known as binary relevance, trains a binary classification problem for each label on data with/without that label. That is, for the jth label, we solve the corresponding jth binary classification problem

$$\boldsymbol{w}_j = \arg\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l} \xi(y_{ij}\boldsymbol{w}^T\boldsymbol{x}_i), \tag{1}$$

where C is a regularization parameter. For the loss function ξ, we support logistic regression and linear SVM through LIBLINEAR (Fan et al., 2008). After the training process ends, for any test instance $\boldsymbol{x}$, the decision function of label j is $f_j(\boldsymbol{x}) = \boldsymbol{w}_j^T\boldsymbol{x}$. Various ways can be applied on decision values to make predictions. The most used method is to use $f_j(\boldsymbol{x})$ as a binary classifier so that

$$\begin{cases} \boldsymbol{x} \text{ is predicted to have the label } j & \text{if } f_j(\boldsymbol{x}) > 0, \\ \text{otherwise} & \text{if } f_j(\boldsymbol{x}) \leq 0. \end{cases} \tag{2}$$

Alternatively, in some applications, labels corresponding to the largest K values of $f_j(\boldsymbol{x}), \forall j$ are predicted to be associated with $\boldsymbol{x}$, where K is a number specified by users.

3.1.2 Thresholding

It is known that under the one-vs-rest setting, for some infrequent labels, the two-class problem (1) is highly imbalanced. Sometimes an instance is predicted to have no labels at all. Thresholding is a technique to address this issue and it is effective to optimize the Macro-F1 score (Lewis et al., 1996; Yang, 1999; Fan and Lin, 2007). The method automatically decides a threshold Δ_j through some cross-validation procedure so that the decision function becomes

$$f_j(\boldsymbol{x}) = \boldsymbol{w}_j^T \boldsymbol{x} + \Delta_j.$$

Therefore, this method is more expensive than one-vs-rest.

3.1.3 Cost-Sensitive

Another scheme to solve the class imbalance problem is cost-sensitive learning, which uses a higher loss on positive training instances. Parambath et al. (2014) give some theoretical support showing that the F1 score can be optimized through cost-sensitive learning. For the label j, they extend problem (1) of one-vs-rest to

$$\boldsymbol{w}_j = \arg\min_{\boldsymbol{w}} \ \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\left(\frac{2-t}{t}\right)\sum_{i:y_{ij}=1} \xi(y_{ij}\boldsymbol{w}^T\boldsymbol{x}_i) + C\sum_{i:y_{ij}=-1} \xi(y_{ij}\boldsymbol{w}^T\boldsymbol{x}_i),$$

where $(2-t)/t$ is the cost of false negatives, and $t \in (0, 1]$. In LibMultiLabel, for each label, a pre-defined grid of (C, t) pairs are checked to find the one leading to the best validation F1 score. The best pair is then applied to the whole training set to get the final decision function of the corresponding label. Therefore, this method is more expensive than one-vs-rest.

3.2 Experiments on the RCV1 Data Set

In multi-label classification, if the number of labels is not large (e.g., 103 classes of the RCV1 set), the setting in (2) instead of the top-K $f_j(\boldsymbol{x})$ values are commonly used for predicting the associated labels of an instance. Results on the test set are then evaluated by a measure such as Micro/Macro-F1.

We begin with checking methods discussed in Section 3.1.

- One-vs-rest
  For example, if l2-loss SVM is considered, we can run

  ```
  python3 main.py --config example_config/rcv1/l2svm.yml
  ```

- Thresholding
  To run this method, we can specify the option `linear_technique` in the configuration file:

  ```
  linear_technique: thresholding
  ```

  Alternatively, we can specify this option in the command line:

  ```
  python3 main.py --config example_config/rcv1/l2svm.yml
                  --linear_technique thresholding
  ```

- Cost-sensitive
  Similarly, we can specify the `linear_technique` in the configuration file or run the following command.

```
python3 main.py --config example_config/rcv1/l2svm.yml
                --linear_technique cost_sensitive
```

We check the performance of neural networks for a comparison with linear classifiers. From Section 2, parameter selection is crucial to neural network models, so we show the results of KimCNN model (Kim, 2014) without/with the tuning procedure.

- KimCNN without parameter selection
  In LibMultiLabel, the following parameter values are considered in the configuration file `kim_cnn.yml`.

```
learning_rate: 0.0005
network_config:
  filter_sizes: [2, 4, 8]
  num_filter_per_size: 128 # filter channels
  activation: relu
  dropout: 0.2
```

  The training command is:

```
python main.py --config example_config/rcv1/kim_cnn.yml
```

- KimCNN with parameter selection
  In LibMultiLabel, a configuration file `rcv1/cnn_tune.yml` is provided for tuning hyper-parameters. It specifies the search range as follows.

```
learning_rate: ['grid_search', [0.001, 0.0005, 0.0001]]
network_config:
  filter_sizes: ['grid_search', [[2,4,8], [4,6]]]
  num_filter_per_size: 128 # filter channels
  activation: relu
  dropout: ['grid_search', [0.2, 0.4, 0.6, 0.8]]
```

  In addition, because we aim to optimize Macro-F1, the following evaluation metric is specified.

```
val_metric: Macro-F1
```

  The running command is

```
python3 search_params.py --config example_config/rcv1/cnn_tune.yml
```

  The tuning procedures returns the best parameter set of learning_rate=0.0005, filter_sizes=[4, 6], and dropout=0.4. The model associated with these parameters are applied to predict the test set.

All experiments were conducted on a computer with an AMD R5950X CPU (for linear classifiers) and a NVIDIA RTX 3090 GPU (for neural networks). Table 2 shows the test performance and the training time.

From Table 2 we see that thresholding and cost-sensitive techniques outperform all other models on Macro-F1. For other metrics, methods based on linear classifiers give competitive results, but take less training time. Therefore, for this data set, there may be no need to use a non-linear method such as neural netowrks.

Table 2: Experiments on the RCV1 data set: linear classifiers versus neural networks

Methods	Macro-F1	Micro-F1	P@1	P@5	Training Time
Linear classifiers					CPU
One-vs-rest	51.89	80.30	95.87	55.77	1 minute
Thresholding	61.48	81.06	95.46	54.32	2.2 minutes
Cost-sensitive	58.08	80.67	95.31	55.51	3.3 minutes
Neural networks (KimCNN)					GPU
w/o parameter selection	47.38	77.21	94.88	54.17	3.2 minutes
w/ parameter selection	50.69	77.43	94.66	54.00	12.7 hours

Table 3: Experiments on the EUR-Lex data set

Methods	Macro-F1	Micro-F1	P@1	P@5	Training Time
Linear classifiers					CPU
One-vs-rest	17.42	52.62	83.47	59.06	12.8 minutes
Thresholding	27.70	56.54	81.94	55.21	2.27 hours
Cost-sensitive	22.04	57.70	80.47	57.69	3.57 hours
Neural networks (BiGRU)					GPU
wo/ parameter selection	20.48	51.56	78.13	52.16	27.8 minutes
w/ parameter selection (grid search)	23.65	59.41	83.65	58.72	24.6 hours
w/ parameter selection (ASHA)	22.70	57.42	82.90	56.38	11.6 hours

3.3 Experiments on the EUR-Lex Data Set

We conduct experiments on the EUR-Lex data set, which has thousands of labels. For such data sets, Macro-F1 is often too low to be used because the F1 scores of many labels are close to zero. Therefore, different from the situation in Section 3.2, the focus now is to check P@K. Currently in LibMultiLabel, we do not provide a configuration file for running linear methods on EUR-Lex. However, one can easily copy `rcv1/l2svm.yml` to the `EUR-Lex` directory and modify the data path/name therein:

```
train_path: data/EUR-Lex/train.svm
test_path: data/EUR-Lex/test.svm
data_name: EUR-Lex
```

By a procedure similar to that in Section 3.2, we report the test performance and the training time in Table 3. Clearly, all three linear-based methods give competitive P@1 but shorter training time than the BiGRU model considered in Section 2 (83.65% and 1 days).

Among the methods of using linear classifiers, it is expected that thresholding and cost-sensitive techniques give better Macro-F1. However, one-vs-rest is the best on the ranking-based metric P@K. The worse ranking of decision values by the other two methods may be because that to optimize Macro-F1, they either adjust the decision function of each label (thresholding) or use different hyper-parameters for obtaining decision functions (cost-sensitive).

3.4 Summary

Our experiments indicate that methods of using linear classifiers are highly competitive for some problems but are cheaper to train. Thus in multi-label text classification, it is recommended that such methods are

applied first to obtain baseline results.

References

R.-E. Fan and C.-J. Lin. A study on threshold selection for multi-label classification. Technical report, Department of Computer Science, National Taiwan University, 2007.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: a library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf`.

Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014. doi: 10.3115/v1/D14-1181.

D. D. Lewis, R. E. Schapire, J. P. Callan, and R. Papka. Training algorithms for linear text classifiers. *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 298–306, 1996.

L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-tzur, M. Hardt, B. Recht, and A. Talwalkar. A system for massively parallel hyperparameter tuning. In *Proceedings of Machine Learning and Systems*, volume 2, pages 230–246, 2020.

R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

J.-J. Liu, T.-H. Yang, S.-A. Chen, and C.-J. Lin. Parameter selection: Why we should pay more attention to it. In *Proceedings of the 59th Annual Meeting of the Association of Computational Linguistics (ACL)*, 2021. URL `https://www.csie.ntu.edu.tw/~cjlin/papers/parameter_selection/acl2021_parameter_selection.pdf`. Short paper.

S. A. P. Parambath, N. Usunier, and Y. Grandvalet. Optimizing F-measures by cost-sensitive classification. In *Advances in Neural Information Processing Systems*, volume 27, 2014.

Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1/2): 69–90, 1999.