Comparative Study on Different Approaches in Optimizing Threshold for Music Auto-Tagging

Khanh Nguyen Cao Minh(✉), Thinh Dang An, Vu Tran Quang, and Van Hoai Tran

Ho Chi Minh City University of Technology, VNU-HCM, Ho Chi Minh City, Vietnam
nguyencaominhkhanh@gmail.com

Abstract. In multi-label classification applied to music auto-tagging, the classification threshold is simply set to a constant value (called static threshold), which is usually unsuitable for the classification on imbalanced datasets. There are many approaches to solve this problem. Some find an appropriate threshold for the whole dataset, while the others find one for each tag or for each individual musical instance. In this paper, we present a method for finding an appropriate classification threshold for each individual track using multiple techniques. The ranking model used to experiment with the thresholding model is built based on fully convolutional neural network structure. The performance of the classifier including the thresholding strategy is evaluated against the classifier using static threshold on various evaluation metrics. The results show that the proposed method helps to improve the classification quality of classifier to testing instances.

Keywords: Multi-label classification · Music tagging
Thresholding strategy

1 Introduction

Tagging is a process of assigning a label (or tag) or multiple labels to a certain object. In digital era, tagging is primarily utilized to tag digital contents. This process is mainly executed by human, based on their inherent knowledge. Music tagging is similar to general tagging, instead it assigns more specific tags in music to audio clips, tracks, or albums. These tags can be genres, instrumentation, rhythmic structure, and so on. Also, the quality of music tagging depends mostly on music knowledge of listeners, or by the intention of composers.

A rising question is "Can music tagging be performed automatically?". An experienced listener can listen to audios and annotate accurately tags to those very quickly. So, it is possible to build a classifier to predict music tags by using audio signal. In this process, the classifier automatically assigns a tag to an

T. K. Dang et al. (Eds.): FDSE 2018, LNCS 11251, pp. 237–250, 2018.
https://doi.org/10.1007/978-3-030-03192-3_18

instance, based on its own characteristics. Music auto-tagging brings us a lot of benefits. For example, it can provide some useful information to listeners, such as genres, instrumentation, rhythmic structure. Moreover, based on characteristics of a listener's favorite tracks, music auto-tagging can help commercial products like Spotify, iTunes to recommend relevant tracks to him, which improves user experience.

To build a classifier, it is necessary to extract features from audio. However, feature extraction process is hard to be conducted because hand-crafted features must be aggregated, so determining which features are good and suitable for tagging is difficult. To overcome this problem, using deep learning is a possible approach. Deep learning has recently become widely used in audio analysis research. The advantage of deep learning is to learn the hierarchical representations of the data. Besides, deep learning scale much better with more data than traditional algorithms. Convolutional neural networks (CNN) [7] represents features in different levels of hierarchy, so the model itself can learn from low-level feature to high-level feature. Especially, many hybrid models are built based on CNN such as convolutional recurrent neural network, fully convolutional network (FCN). FCN is built on CNN but replaces the last layer by a convolutional layer instead of a fully connected one. Therefore, FCN maximizes the advantage of convolutional networks. It not only reduces the number of parameters by sharing weights but also aggregates features. Nevertheless, the biggest problem of deep learning is about the availability of labeled datasets. Recently, many large-scale music datasets are released, such as Million Song Dataset (MSD) [1], MagnaTagATune (MTAT) [6], Free Music Archive (FMA) [4]. These datasets are used in many studies because of their quality. Another challenge is the noise in the datasets.

In single-label classification, the label that has the best score can be chosen to be the output, while in multi-label classification, many labels can be chosen to form the result. Area under the curve (AUC) value is often used as a reliable metric to reflect how well a multi-label classifier performs. Recently, many researches only focus on AUC to evaluate the performance of classifiers because of the accuracy paradox. However, if AUC is used as the only evaluation metric, some limitation still remains. Specifically, high value in AUC does not assure high value in accuracy. Currently, most of classification models use a same static threshold for all testing instances and all labels. AUC is not affected by this threshold, but accuracy is. Some tracks are hard to be classified to specific tags (labels) because the scores of all tags are not higher than the static threshold, though there is at least one score higher than the others. Therefore, the classifier predicts that those tracks do not belong to any tag. This leads to a significant decline in value of accuracy.

The first approach to think of is based on scores in validation set to configure a static threshold for all track instances. For example, it can be done by performing a brute-force search for the best of threshold t under a given criterion. t will then be applied to the scores produced for verification set to evaluate the overall performance of the classifier. However, this approach has two problems. First, the

score distribution on validation set does not represent the counter part in testing data. For example, although the score distribution on validation set (Fig. 1a) and that on verification set (Fig. 1b) show many similarities, there are still differences between them in detail. Second, all scores of a specific track can be lower than the static threshold, which results in no label belonging to this track, although one of its scores is much higher than the others. So, assigning a constant threshold to all instances is not appropriate.

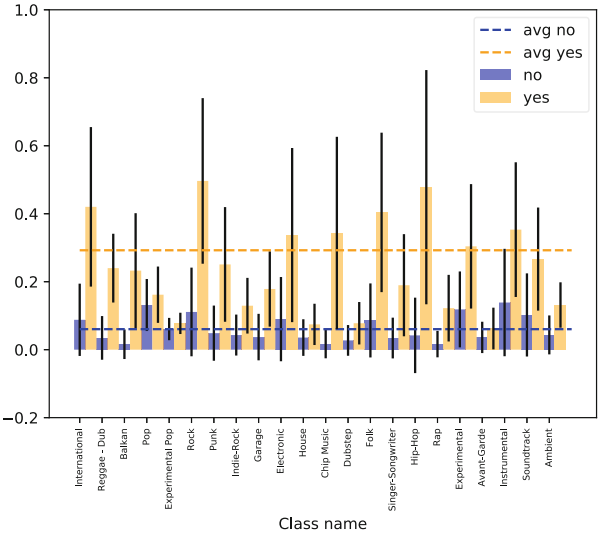
There are researches that propose algorithms to generate a reasonable threshold for each instance instead of using a static one. RCUT [8] generates a threshold to guarantee that a fixed proportion of tags is predicted as positive, which is not common with music tagging problems where a number of tags assigned to each instance are not constant. Beside RCUT, MCUT [8] is a wonderful method to produce the threshold, it splits positive predictions and negative ones in a position where two scores have the largest distance. In general, it plays an important role to choose an appropriate threshold based on the characteristics of each instance. There are various metrics to assess this threshold. Among them, F_β [9] is the most suitable, with β is used to adjust weight between precision and recall.

The basic idea of above researches is to separate the classifier into two individual parts, the ranking model and the thresholding model. For each musical instance, the ranking model generates scores of all tags whereas the thresholding model generates an appropriate threshold value. The overall result of the classifier recommends tags whose scores are higher or equal to the generated threshold. However, the approaches mentioned above do not perform well in practice. There are two main reasons for this. First, they are hard-coded (for RCUT it is the fixed proportion of tags that are predicted as positive). Second, they do not utilize the features of each instance. To deal with these limitations, we propose a method to implement a thresholding model which can learn from labeled instances and then generates a good threshold for testing ones based on their characteristics.

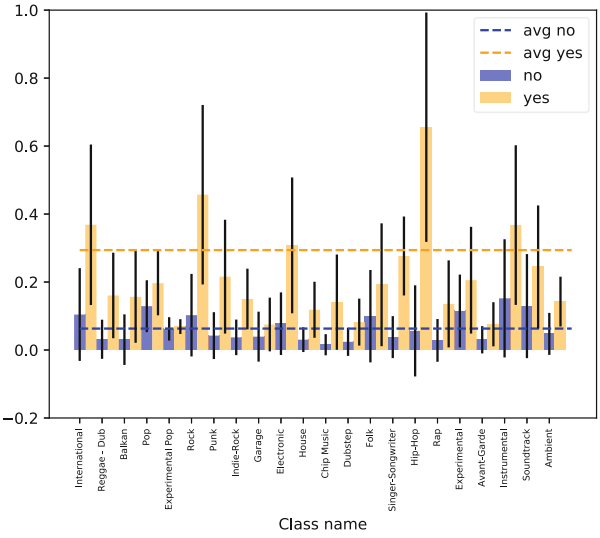
The rest of the paper is organized as follows. Section 2 mentions researches working on determining an appropriate threshold value, their pros and cons. Section 3 describes our proposed method in details while Sect. 4 presents datasets used in this study, the structure of experimented models, the optimization and the evaluation metrics as well as experiment results. Finally, Sect. 5 summarizes the achievement of the research.

2 Related Work

Currently, all of the work uses ranker's score, which is output of the ranking model, as input to the threshold generator. Some algorithms require training or optimizing on training set before deployment. The training set may be the validation set of ranker, or can be created by cross validation when training ranker on its training set. That training set has the same label set as ranker's dataset but its instance set is ranker's scores.



(a) On validation set



(b) On verification set

Fig. 1. Mean and standard deviation of score by genre. Yellow color shows the set of instances having that genre, while blue shows set of instances that do not. (Color figure online)

The study [8] provides basic method for threshold selection. SCut finds threshold to minimize specific loss function, TCut finds threshold to guarantee the ratio of positive prediction on training set. TCut and SCut are label-level thresholding algorithms while RCut, MCut are instance-level thresholding ones. RCut returns the threshold that lets classifier predict k labels for each individual instance. For MCut, label scores are sorted first, resulting threshold is the average value of two consecutive scores, whose gap is maximum.

Some works research on label-based thresholding [5,10]. For each class, look for the threshold that optimize specific label-level classification metric. [5] provides method for mapping from threshold optimized in training set to verification set, but this method will not work on different instances. In general, all of the optimization methods above tend to be overfit on training set.

Machine learning is the statistical method that work well on testing data by optimizing on training set. For each instance, the proposed method generates instance-level threshold that optimize specific instance-level classification metric, this method involves using k-nearest neighbors (KNN) [2].

In this paper, besides machine learning technique (in particular, KNN), we propose another instance-based approach to determine threshold that uses deep learning and transfer learning. Moreover, our model is optimized using various metrics Subsect. 4.2. Finally, we experiment our method on two popular large-scale music dataset MSD and MTAT, then compare it with machine learning one and classifier using static threshold.

3 Different Approaches in Determining Instance-Based Threshold

As mentioned above, the ranking model f takes its input as a waveform x of a track. Its output is a score vector $s \in [0,1]^q$ of that track. However, the main purpose is to classify tags, which means the expected output is a binary vector $p \in \{0,1\}^q$. Each element of p indicates the predicted classification of a certain tag. If $p_i = 0$, that track does not belong to tag i, and vice versa for $p_i = 1$. So, it is necessary to have a thresholding model l whose output is a threshold value t. If the score s_i of a certain tag i is higher than t, $p_i = 1$. And if not, $p_i = 0$.

To implement l, we propose an instance-based thresholding model. This means, instead of estimating a static threshold for all instances or all labels, the instance-based method will generate threshold value for each particular instance x. The value is used to decide the classification result p. Our proposed method to implement the instance-based thresholding model is described as follows.

3.1 Splitting the Dataset

Note that, the proposed classifier consists of two individual parts - the ranking model and the thresholding model. They both need to be trained, so it is necessary to split the dataset appropriately for our usage. Given the dataset is $D = \{(x, y)\} = D^{Tr} \cup D^{Va} \cup D^{Ve}$, with $x \in \mathbb{R}^d$ is waveform of track instance

and $y \in [0,1]^q$ is the corresponding score vector of all tags. Because y is the true label of x, so each of its element is standardized to 0 or 1, which guarantees that the track belongs or does not belong to a certain tag. Since the ranker is trained on the training set $D^{Tr} = \{(x^{Tr}, y^{Tr})\}$, it is not used to train the thresholding model. Neither is the verification set $D^{Ve} = \{(x^{Ve}, y^{Ve})\}$ because it is used to evaluate the performance of both model. Validation set $D^{Va} = \{(x^{Va}, y^{Va})\}$ is the last resort. D^{Va} is used to generate $D_t = D_t^{Tr} \cup D_t^{Va}$, which are used to train and validate the thresholding model, respectively. Generating method of D_t will be described later.

3.2 Building and Training the Ranking Model

The ranking model f is built and trained on the training set D^{Tr}, detailed block diagram of the model is shown in Fig. 2.

The ranking model is a convolutional neural network which is designed based on [3] (Fig. 2). Its input is the waveform which is then transformed to a spectrogram. The spectrogram has a fixed size $96 \times 1366 \times 1$ corresponding to frequency, time and channel dimension, respectively. It goes through batch normalization layer along the frequency axis. Then, the output of this layer is put through five convolutional blocks. Each block consists of a 2-D convolutional layer to extract features, a batch normalization layer along the channel axis to stabilize the value domain, followed by a ReLU activation layer, a dropout layer with ratio 0.2 to prevent overfit and a max pooling layer at last to aggregate features and reduce feature map size. The output of these blocks are vectorized and connected to a sigmoid dense layer whose number of neurons is equal to the number of tags. The loss function is binary cross entropy.

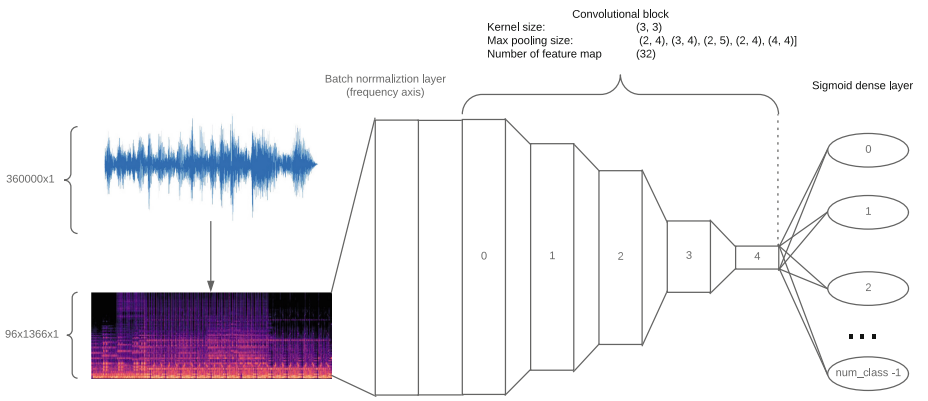


Fig. 2. Block diagram of the ranking model [3]: Audio of a track is first preprocessed to create its corresponding mel-spectrogram. This image is the input of the ranking model whose first layer is the batch normalization layer along the frequency axis, followed by five convolutional blocks. The last layer is a sigmoid dense layer.

3.3 Generating the Dataset for the Thresholding Model

The dataset D_t is used to train and validate the thresholding model. Label set of D_t is formed by using Algorithm 1 called g_m. The purpose of this algorithm is to generate an appropriate threshold that optimizes a certain instance-based metric. For each instance x^{Va} in D^{Va}, the trained ranker f^{Tr} from step 2 will predict its corresponding score vector s^{Va}. Given s^{sorted} is the descending-sorted version of s^{Va}, sequentially use each score e in s^{sorted} as the classification threshold. Consequently, the chosen label y_t for the dataset of the thresholding model l_m that optimizes metric m is the mean of two values. The first is the score $s_{i_m}^{sorted}$ that makes the predicted classification p^{Va} achieve highest $m(p^{Va}, y^{Va})$. The second is $s_{i_m+1}^{sorted}$. In general, the above algorithm g_m needs two parameters, instance x and its true label y, to generate the threshold $y_t = gm(x, y)$. When deployed, l_m is not provided with the true label $\hat{y}$ of testing instance $\hat{x}$, so it needs to learn how to predict the best threshold $\hat{y}_t$ that optimizes metric m for $\hat{x}$.

3.4 Designing the Thresholding Model

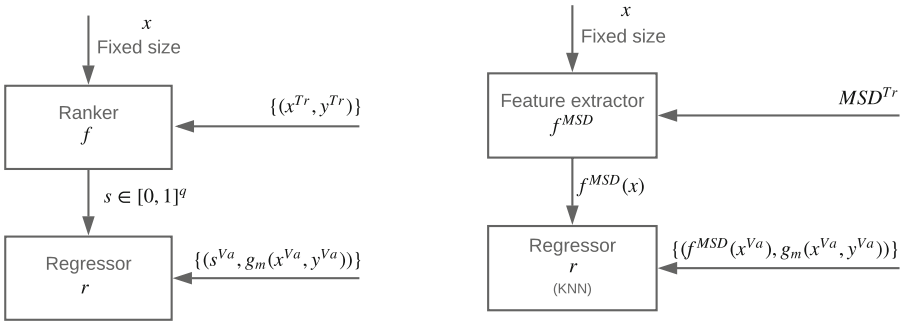
Building the thresholding model, training and validating it on D_t^{Tr} and D_t^{Va}, respectively. The thresholding generating problem is a regression one, so it is necessary to design a regressor r. The paper proposes five approaches to solve it. All of these are described as follows.

1. KNN: The regressor uses k-nearest neighbors regression with $k = \sqrt{|X_t^{Tr}|}$, $|X_t^{Tr}|$ is the number of instances in D_t^{Tr}. For each instance in verification data, r will calculate its similarity to the training instances by certain distance metric (e.g. Euclidean distance). The mean of threshold values of k closest instances is set as the threshold of verification instance, with k is the parameter of KNN. Its dataset consists of pairs (x_t, y_t), with x_t is the score vector $s^{Va} = f(x^{Va})$ of instance $x^{Va} \in D^{Va}$ and $y_t = g_m(x^{Va}, y^{Va})$. So, the dataset for the regressor r is $\{(s^{Va}, y_t)\}$.
2. KNNS: Similar to KNN, but includes sorting the input score vector in descendant. Because Algorithm 1 sorts scores to generate labels for the dataset, it is expected that adding the sorting step will help the model determine the threshold better.
3. MLP: Deep learning regressor is built based on multi-layer perceptron structure. The used structure consists of two fully-connected layers, each has 50 neurons. The loss function is the mean square error (MSE) between the predicted threshold t and the true label y_t. r is trained and validated on the same dataset as KNN and KNNS. For each instance in test set, r will predict the corresponding threshold value.
4. MLPS: Similar to MLP with the sorting step added.
5. TL: This approach uses transfer learning technique. While the instance set of above approaches is the score vector s^{Va} of $x^{Va} \in D^{Va}$, the instance set of TL is the learned features of the track extracted from the ranker that was

trained on MSD dataset (f^{MSD}) [3]. So, for this approach only, the dataset for the regressor is $\{(f^{MSD}(x^{Va}), y_t)\}$. The classification algorithm used for TL is k-nearest neighbors regression.

6. T0.5: The thresholding model that always generates a threshold value 0.5 for all music instances is also experimented as a baseline to compare with the five mentioned approaches.

Figure 3 visualizes how the data is used in five proposed approaches.



(a) Methods (a) - (d) (b) Method (e)

Fig. 3. Data usage in two training method classes of thresholding models l. Figure 3a describes methods 1–4. Figure 3a describes method 5. Horizontal arrows show data used in training while vertical arrows show data in verification. In all five approaches, the ranker f is trained on D^{Tr}, and MSD^{Tr} specifically in approach TL. For the regressor r, approaches 1–4 use the dataset containing pairs of score vector as instance and threshold value as label. Approach TL uses the dataset containing pairs of extracted feature from f^{MSD} as instance and threshold value as label.

4 Experiment

4.1 Dataset

To train and evaluate performance of the proposed method, we use three datasets Million Song Dataset (MSD) [1], MagnaTagATune (MTAT) [6] and Free Music Archive (FMA) [4]. These datasets appear to be well suited for training Deep learning ranking model because of their large collection of tracks and tags.

While it is very easy to download both data and metadata of MTAT and FMA dataset, MSD forces researchers to download audio clips from online services. But thanks to Mr. Keunwoo Choi[1], we managed to get almost all data of MSD. We really appreciate his enthusiastic help.

For MSD, we split the dataset into training/validation/test set based on convention proposed in[2]. Specifically, only track instances that have tags in

[1] https://keunwoochoi.wordpress.com/.
[2] https://github.com/keunwoochoi/MSD_split_for_tagging.

Input: An instance x
A label vector y that associates with x
A ranking model f that was trained on training set D^{Tr} and can generate score $s = f(x)$ as a vector for instance x
An instance-based metric m, assumption of m is that greater is better
Output: A threshold y_t that optimizes metric m
$s \leftarrow f(x)$;
$s^{sorted} \leftarrow sort\_descending(s)$;
Initialize M as empty list to save metric result $m(y, p)$;
for $i \leftarrow 1$ **to** $size(s^{sorted})$ **do**
  $e = s_i^{sorted}$;
  `// Compute bit vector p indicate the prediction after applying threshold` e `to score vector` s
  $p \leftarrow unit(s, e)$;
  $M.append(m(y, p))$;
end
`// Find index of maximum metric result`
$i_m \leftarrow index\_max(M)$;
$y_t = average(s_{i_m}^{sorted}, s_{i_m+1}^{sorted})$;
return y_t;

Algorithm 1. Generate labels for the dataset used to train and evaluate the thresholding model.

50 most popular tags are kept. After this process, training set consists of 201680 instances, validation set 12605 instances and verification set 25904 instances.

For MTAT, this dataset has a problem of synonymous tags, so it needs to be preprocessed based on[3]. Then, MTAT is split into training/validation/verification set with ratio 12/1/3. Similar to MSD, only track instances that have tags in 50 most popular tags are kept.

For FMA, it has three versions - large, medium and small. Based on its small version which consists of 8 genres, 8000 track instances, 1000 per genre, we generate a new version called FMAS2 that includes some children of these 8 genres. The reason behind this is to support the multi-label classification problem. Consequently, FMAS2 consists of 22 genres and 8000 track instances. Then, it is splited into training/validation/verification set with ratio 8:1:1.

4.2 Optimization and Evaluation Measures

The proposed method develops multiple thresholding models, each optimizes a certain instance-based metric. In particular, these metrics are *Accuracy*, *Precision*, *Recall* and F_β. They are defined as follows:

- $Accuracy_{inst} = \frac{TP}{TP+FP+FN}$,
- $Precision_{inst} = \frac{TP}{TP+FP}$,

[3] https://github.com/keunwoochoi/magnatagatune-list.

- $Recall_{inst} = \frac{TP}{TP+FN}$,
- $F^{\beta}_{inst} = \frac{(1+\beta^2)Precision_{inst}Recall_{inst}}{\beta^2 Precision_{inst}+Recall_{inst}} = \frac{(1+\beta^2)TP}{(1+\beta^2)TP+\beta^2 FN+FP}$,

with TP, FP, TN and FN are True Positive, False Positive, True Negative and False Negative, respectively. β is a parameter used to adjust the weight between $Precision$ and $Recall$. In our experiment, we use $\beta = 0.5, 1, 2$.

For evaluation, above metrics are used but calculated by two ways, instance-based and label-based. Additionally, $SubsetAccuracy$ is also calculated. Moreover, they are averaged to evaluate the performance of the thresholding model on the whole testing set. Given the number of instances in testing set is D and the number of labels is q, their formulas need to redefined as below:

- $\overline{Accuracy_{inst}} = \frac{1}{D}\sum_{i=1}^{D}\frac{TP_i}{TP_i+FP_i+FN_i}$,
- $\overline{Precision_{inst}} = \frac{1}{D}\sum_{i=1}^{D}\frac{TP_i}{TP_i+FP_i}$,
- $\overline{Recall_{inst}} = \frac{1}{D}\sum_{i=1}^{D}\frac{TP_i}{TP_i+FN_i}$,
- $\overline{F^{\beta}_{inst}} = \frac{1}{D}\sum_{i=1}^{D}\frac{(1+\beta^2)TP_i}{(1+\beta^2)TP_i+\beta^2 FN_i+FP_i}$,
- $\overline{Accuracy_{label}} = \frac{1}{q}\sum_{j=1}^{q}\frac{TP_j+TN_j}{TP_j+FP_j+TN_j+FN_j}$,
- $\overline{Precision_{label}} = \frac{1}{q}\sum_{j=1}^{q}\frac{TP_j}{TP_j+FP_j}$,
- $\overline{Recall_{label}} = \frac{1}{q}\sum_{j=1}^{q}\frac{TP_j}{TP_j+FN_j}$,
- $\overline{F^{\beta}_{label}} = \frac{1}{q}\sum_{j=1}^{q}\frac{(1+\beta^2)TP_j}{(1+\beta^2)TP_j+\beta^2 FN_j+FP_j}$,
- $\overline{subset\_acc} = \frac{1}{D}\sum_{i=1}^{D}(p_i = y_i)$.

4.3 Experiment Result

To evaluate the performance of the thresholding model, we use the evaluation metrics mentioned in Subsect. 4.2. Table 1 shows the overall performance of five proposed approaches when trained and verified on MSD, MTAT, FMAS2 datasets. The experimented thresholding models $l_{F^1_{inst}}$ focus on optimizing F^1_{inst} and the evaluated metric is $\overline{F^1_{label}}$. All five approaches give equivalent result and much better than that of T0.5. KNN, KNNS, MLP and MLPS show the best result, though KNNS and MLPS need more computational cost because of the sorting step. Moreover, the bigger the dataset is, the more slowly KNN and KNNS execute. The computational cost of transfer learning approach is much higher than that of k-nearest neighbor ones as the size of its optimal input feature vector is 64, whereas the score vector has fewer elements. In conclusion, multi-layer perceptron without sorting step approach is an appropriate choice to deploy. The rest of this section focuses on analyzing the MLP thresholding model $l_{F^1_{inst}}$.

Looking at Fig. 4, the thresholding model that optimizes $Recall_{inst}$ always generates threshold value 0, which is as expected. With this value, recall metrics are always equal to 1. So, this model is useless. We just experiment it to verify that our proposed method work well. However, the other five models generate meaningful threshold values. The standard deviation is about 0.15.

Table 1. $\overline{F^1_{label}}$ of classifiers using proposed thresholding strategies on datasets MSD, MTAT, FMAS2. All thresholding models optimize F^1_{inst} metric. The gray cell indicates that approach TL is not experimented on MSD dataset.

Classifier	MSD dataset	MTAT dataset	FMAS2 dataset
T0.5	0.0426	0.2450	0.1385
KNN	0.1977	0.3424	0.2446
MLP	0.1912	0.3340	0.2337
KNNS	0.1948	0.3371	0.2341
MLPS	0.1914	0.3422	0.2480
TL		0.3401	0.2529

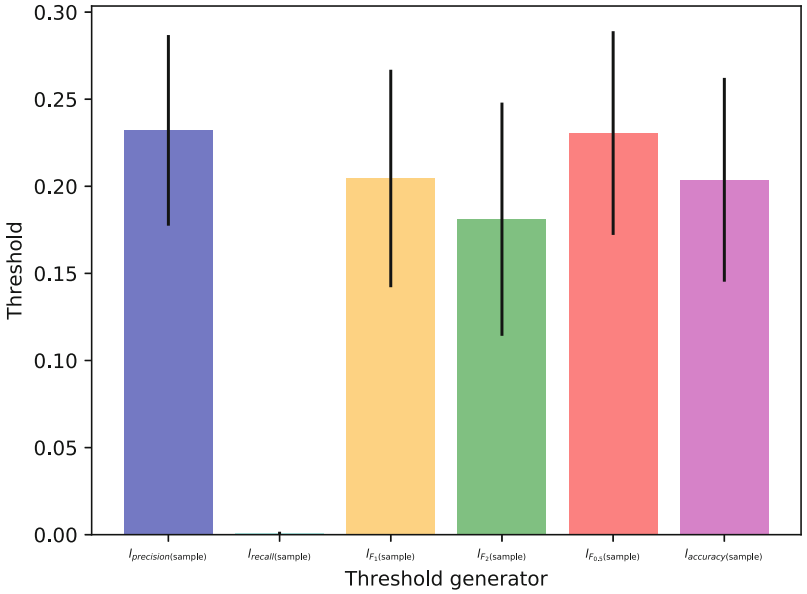
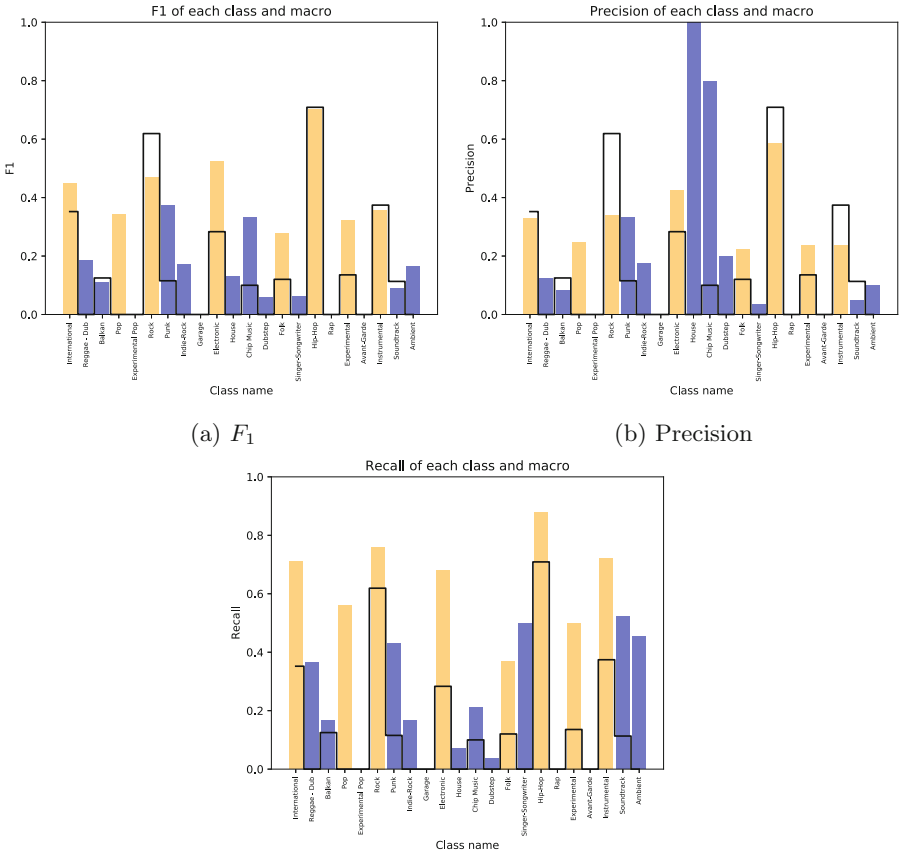


Fig. 4. Mean and standard deviation of the generated threshold from each thresholding model l_m. There are six models optimizing different instance-level measures m. They generate threshold value y_t for each instance.

The threshold values lie between 0.15 and 0.3. Referring back to Fig. 1, the score mean of instances that are predicted to have a tag is around 0.3, whereas the score mean of instances that are not is around 0.07. So, it is reasonable that the generated threshold is valued from 0.15 to 0.3. The thresholding models that prioritize recall generate lower threshold values for the same reason as $l_{Recall_{inst}}$. $l_{Precision_{inst}}$ generates highest threshold, followed by $l_{F^{0.5}_{inst}}$, $l_{F^1_{inst}}$, $l_{F^2_{inst}}$, $l_{Recall_{inst}}$.

It is obvious that $\overline{F^1_{label}}$ improves when using MLP model that optimizes F^1_{inst} compared to T0.5 (Fig. 6). However, for each tag, not every F^1_{label} improves



(a) F_1 (b) Precision

(c) Recall

Fig. 5. Value of F_1, precision, recall on verification set of FMAS2 by genre. The classifier is evaluated using the MLP thresholding model $l_{F^1_{inst}}$ that optimizes F^1_{inst}. Orange columns show parents of tags that are demonstrated by right blue columns. (Color figure online)

(Fig. 5a). Specifically, only F^1_{label} of rock reduces significantly; balkan, hip-hop, instrumental and soundtrack show the same value as their corresponding baseline, the other 16 tags improve substantially compared to the baseline. F^1_{label} is the harmonic mean of $Precision_{label}$ and $Recall_{label}$. Based on Fig. 5c, all recall values are increased, so the model has precision values reduced (Fig. 5b). Since $l_{F^1_{inst}}$ generates average threshold value less than 0.5 (Fig. 4), the classifier produces more positive predictions, which leads to the increase in recall. However, the number of false positives increasing in some tags results in the reduction in their corresponding precision.

Figure 6 visualizes results of many metrics on verification set of FMAS2 when using five models optimizing $Precision_{label}$, F^1_{inst}, F^2_{inst}, $F^{0.5}_{inst}$ and $Accuracy_{inst}$.

All these five perform significantly better than the baseline. Specifically, all evaluation results are increased except for $\overline{Precision_{label}}$, $\overline{Accuracy_{label}}$, $\overline{subset\_acc}$. The reason why $\overline{Accuracy_{label}}$ and $\overline{subset\_acc}$ decrease is because all optimization metrics do not care about true negative, while these two have true negative in their formulas. Additionally, optimizing instance-level metrics also help to increase result of label-level metrics. However, label-level metrics show less improvement than instance-level ones. Precision is an exception. For baseline T0.5, $\overline{Precision_{label}}$ is higher than $\overline{Precision_{inst}}$. While for the thresholding models, $\overline{Precision_{label}}$ is much lower than $\overline{Precision_{inst}}$, even lower than baseline's $\overline{Precision_{label}}$.

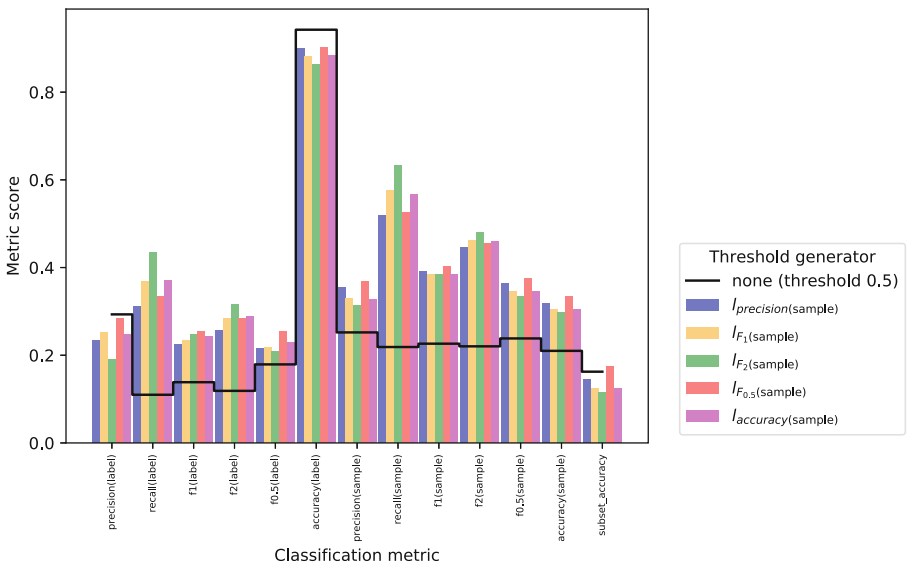


Fig. 6. Performance evaluation of the classifier with five different thresholding models. Horizontal axis consists of evaluation metrics whereas vertical axis shows performance value of each metric.

5 Conclusion

From Sect. 4, it is obvious that the classifier combined with a thresholding model achieves better performance than that using a static threshold. Through experiment, it is an appropriate choice to use multi-layer perceptron structure without sorting step to deploy a thresholding model in real projects. The improvement in the performance of the classifier gained from experiment is also significant. However, there is some aspect to consider that can help the improvement even better. A recommendation to further enhance the performance of the thresholding models is described as follows. Beside determining the target threshold based on only each individual instance, it is possible to put label-based thresholding

method together with instance-based one. Consequently, for each instance, the thresholding model generates a vector, each value in that vector demonstrates the threshold for a label in label set. With this approach, instances are independent of each other, as well as labels.

Acknowledgements. The authors would like to thank Faculty of Computer Science and Engineering, HCMC University of Technology for providing computing facilities to this study. The experiments presented in this paper are tested on the High Performance Computing Lab (HPC Lab) of the faculty.

References

1. Bertin-Mahieux, T., et al.: The million song dataset. In: Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011) (2011)
2. Chen, B., Gu, W., Hu, J.: An improved multi-label classification method and its application to functional genomics. Int. J. Comput. Biol. Drug Des. (IJCBDD) **3**(2), 133–145 (2010)
3. Choi, K., et al.: Transfer learning for music classification and regression tasks. CoRR abs/1703.09179 (2017). arXiv:1703.09179
4. Defferrard, M., et al.: FMA: a dataset for music analysis. In: 18th International Society for Music Information Retrieval Conference (2017). https://arxiv.org/abs/1612.01840
5. Fan, R.-E., Lin, C.-J.: A Study on Threshold Selection for Multi-label Classification. Department of Computer Science, National Taiwan University (2005)
6. Law, E., et al.: Evaluation of algorithms using games: the case of music tagging. In: Proceedings of the 10th International Society for Music Information Retrieval Conference, ISMIR 2009, pp. 387–392 (2009). ISBN 9780981353708. English (US)
7. LeCun, Y., Haffner, P., Bottou, L., Bengio, Y.: Object recognition with gradient-based learning. Shape, Contour and Grouping in Computer Vision. LNCS, vol. 1681, pp. 319–345. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-46805-6_19
8. Al-Otaibi, R., Flach, P., Kull, M.: Multi-label classification: a comparative study on threshold selection methods (2014)
9. Zhang, M., Zhou, Z.: A review on multi-label learning algorithms. IEEE Trans. Knowl. Data Eng. **26**(8), 1819–1837 (2014). https://doi.org/10.1109/TKDE.2013.39. ISSN: 1041-4347
10. Zou, Q.: Finding the best classification threshold in imbalanced classification. Big Data Res. **5**, 2–8 (2016)