A Non-Linear Label Compression and Transformation Method for Multi-Label Classification using Autoencoders - Supplement

Jörg Wicker, Andrey Tyukin, and Stefan Kramer

Institut of Computer Science, Johannes Gutenberg University Mainz
Staudingerweg 9, 55128 Mainz, Germany
wicker@uni-mainz.de, tyukinandrey@gmail.com, kramer@informatik.uni-mainz.de

1 Maniac – Multi-Label Classification using Autoencoders

The algorithm works similarly to other label compression based algorithms like MLC-BMaD [1] or the method proposed by Tai and Lin [2]. In the first step, the labels are compressed using a compression algorithm. Then a base learner or a set of base learners is trained on the compressed labels (see Figure (1)). In the case of Maniac, we use autoencoders for compression, in the case of MLC-BMaD matrix decomposition is used, Tai and Lin use singular value decomposition. The difference in using the autoencoders in this step is that unlike other approaches, this captures non-linear dependencies among the labels. Boolean matrix decomposition and SVD cannot capture them, hence the performance regarding the prediction should be improved on data sets with non-linear dependencies (experiment shown in main paper).

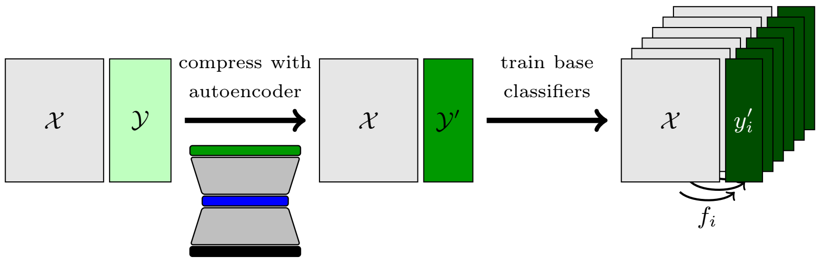


Fig. 1: Visualization of the training phase of Maniac. After the autoencoders are trained (not shown), $\mathcal{Y}$ is compressed using the trained autoencoders. Next, a BR model is trained on the compressed labels.

Autoencoders [3] are neural networks that consist of a compressor and a decompressor part, connected by a small central layer (see Figure (2)). In the following, we will explain details of the autoencoder training.

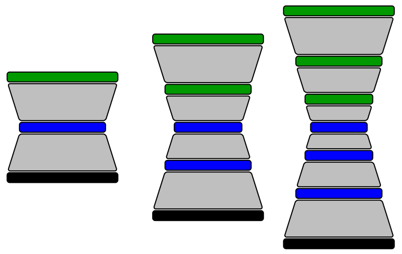


Fig. 2: Stream of increasingly deep autoencoders. The deeper autoencoders are created from the shallower ones by unfolding the innermost layer and then tuning with backpropagation.

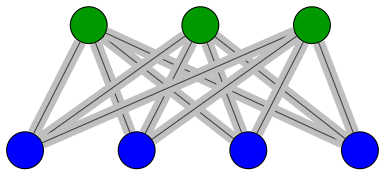


Fig. 3: Topology of a Restricted Boltzmann Machine (RBM). Nodes in the lower layer (blue) are called *visible units*. Nodes in the upper layer (green) are called *hidden units*.

Autoencoders can be trained in an unsupervised manner[1]. For this, training data is clamped to the equally sized input and output layers of the autoencoder. The training data is compressed using the compressor part of the network, and then reconstructed again using the decompressor part. The result is compared to the original data, the reconstruction errors are used to tune the parameters of the compressor and the decompressor. More specifically, we used a Conjugate Gradient (CG) algorithm, and computed the necessary gradients from reconstruction errors using classical backpropagation. If the training of an autoencoder is successful, the output of the network is similar to the input data. Since the output is reconstructed by the decompressor from the activations of neurons in the innermost layer, the small innermost layer of the autoencoder can be thought of as an informational bottleneck. The activations of the neurons in this bottleneck layer constitute an efficient low-dimensional representation of the input data. In particular, this representation captures non-linear dependencies between the activations of the neurons in the input layer.

The naïve approach to train autoencoders does not work very well: If one initializes the neurons randomly, and then trains all layers at once using only backpropagation, then the training takes unreasonably long time, and the resulting model is rather poor. One has to either adjust the training procedure or come up with a more sophisticated method to initialize the neurons before the actual backpropagation-based training begins.

Hinton and Salakhutdinov [3] proposed to treat pairs of layers (see Figure (3)) of the autoencoder as *Restricted Boltzmann Machines* (RBMs), and train them using the *Contrastive Divergence* (CD) algorithm. Connection weights and

[1] It should be noted that, as we train the autoencoders on the labels, this could be understood as supervised learning. Nevertheless, for the training of the autoencoders, no additional target variable is used, and the labels are not treated as target variables for this step, hence this is still unsupervised training.

neuron activation biases produced by contrastive divergence were then used as input to the backpropagation-based optimization algorithm.

We use a variation of these ideas to train whole *streams* of increasingly deep autoencoders. The training process of autoencoders is as follows. We start with a trivial autoencoder that has a single neuron layer and no connections. Suppose that we already have an autoencoder with $2n+1$ layers. We apply the compressor part to the data, and train a new RBM on the compressed data. The size of the new layer of hidden units (see Figure (3)) is determined by the *compression factor* metaparameter. Then we unfold the RBM and merge it into the center of the original autoencoder, obtaining an autoencoder with $2n+3$ layers (see Figure (2)). Hinton and Salakhutdinov originally proposed to keep unfolding all RBMs until the desired depth is reached, and fine-tune the final autoencoder with the backpropagation algorithm in the very end. However, we have found it beneficial to treat the unfolded RBMs as small autoencoders of depth 1 and also tune them with backpropagation. After gluing the new small autoencoder into the center of the previously obtained $(2n+1)$-layer autoencoder, we also tune the resulting $(2n+3)$-layer autoencoder with backpropagation.

We do not repeat the description of the backpropagation or contrastive divergence, since it is the same as proposed by Hinton and Salakhutdinov [3]. However, we should explain what exactly we mean by "gluing" small autoencoders into larger ones.

Let for a moment denote the compressor part (input layer to the central bottleneck layer) by c, the remaining decompressor part using d, and the input data with $x \in \mathbb{R}^{N \times D'}$. Here, N is the number of instances and D' the dimension of the compressed data. First, we compress the data using the compressor part d and obtain the neuron activations in the central bottleneck layer:

$$y := c(x) \in \mathbb{R}^{N \times D'}.$$

For each column index $i = 1 \dots D'$ we compute coefficients of an affine linear transform $\xi \mapsto a_i \xi + b_i$ such that for each i it holds:

$$\min_{j \in 1 \dots N} a_i y_{ji} + b_i = 0 \qquad \max_{j \in 1 \dots N} a_i y_{ji} + b_i = 1.$$

For brevity, write $A := \operatorname{diag}((a_i)_i)$, $b := (b_i)_i$, $\psi(y) := Ay + b$.

Now we can use this affine linear transform to make the compressed data y suitable for training of a new RBM. Since RBMs interpret the input as probabilities,

$$z := \psi(y) \in [0,1]^{N \times D'}$$

is suitable training data for an RBM. Now we train a new RBM with the visible layer of size D', and hidden layer of size $D'' := \mathit{compressionFactor} \cdot D'$. Let $h \in \mathbb{R}^{D''}$, $v \in \mathbb{R}^{D'}$ and $W \in \mathbb{R}^{D'' \times D'}$ denote the resulting hidden and visible layer biases and connection weights, respectively.

Now we can unfold the RBM into a small autoencoder as shown in the figure (4), and fine tune it using usual backpropagation. Let $\tilde{v}$, $\tilde{h}$, $\tilde{W}$ and $\tilde{W}^{\dagger}$ denote biases and connection weights resulting from the backpropagation.

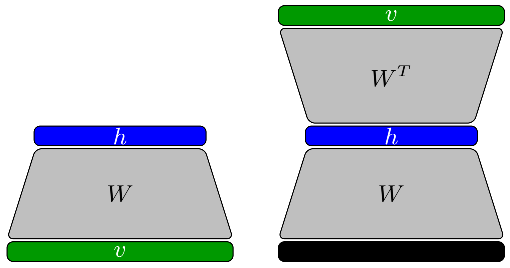


Fig. 4: On the left: an RBM with visible units v, hidden units h and connection weights W. On the right: an RBM unfolded into a small autoencoder. The transpose matrix W^T is used as connection between the bottleneck layer and the output layer. The black box at the bottom is the parameter-less input layer.

We now want to insert the new fine-tuned small autoencoder between the compressor part c and the decompressor part d of the larger autoencoder, and to use ψ and ψ^{-1} as glue. The new autoencoder should implement the following function:

$$\mathcal{A}' := d \circ \psi^{-1} \circ \sigma_{\tilde{v}} \circ \tilde{W}^\dagger \circ \sigma_{\tilde{h}} \circ \tilde{W} \circ \psi \circ c. \tag{1}$$

Here we denote linear functions by the same symbols as the corresponding matrices in the standard basis, and use σ for the component-wise sigmoid function:

$$\sigma((\xi_i)_i) := \left(\frac{1}{1 - e^{-\xi_i}}\right)_i \qquad \sigma_b(\xi) := \sigma(b + \xi).$$

The rationale for building a new autoencoder like this is as follows: if we assume that the following three (approximate) equations hold:

$$d \circ c \approx Id \qquad \psi^{-1} \circ \psi = Id \qquad \sigma_{\tilde{v}} \circ \tilde{W}^\dagger \circ \sigma_{\tilde{h}} \circ \tilde{W} \approx Id,$$

then it is also reasonable to assume that $\mathcal{A}'$ will behave approximately like an identity function, but having a (lower dimensional) neuron layer $\sigma_{\tilde{h}}$ as bottleneck.

At first glance, it is not immediately obvious that the function (1) can still be described by the classical neural network architecture with alternating connection and neuron layers. In order to express the function (1) as such an neural network, we have to somehow eliminate the affine linear transforms ψ and ψ^{-1}. Fortunately, the linear transformation part can be hidden in the connection layers, and the offsets can be hidden in biases of neuron layers as follows. Consider the composition $(\sigma_{\tilde{h}} \circ \tilde{W} \circ \psi)$. For all inputs ξ of suitable dimension, it holds:

$$\begin{aligned}(\sigma_{\tilde{h}} \circ \tilde{W} \circ \psi)(\xi) &= \sigma_{\tilde{h}}(\tilde{W}(A\xi + b)) \\ &= \sigma(\tilde{W}(A\xi + b) + \tilde{h}) \\ &= \sigma(\tilde{W}A\xi + \tilde{W}b + \tilde{h}) \\ &= (\sigma_{\tilde{h}+\tilde{W}b} \circ \tilde{W} \circ A)(\xi).\end{aligned} \tag{2}$$

Essentially the same can be done with ψ^{-1}, but for this, we have to actually take the decompressor d apart, and modify some of its components. For this, suppose that $d = g \circ \sigma_q \circ M$, where M is a connection matrix, σ_q stands for a layer of neurons, and $\tilde{g}$ is some composition of alternating connections and neuron layers is the rest of the decompressor. It then holds (for a suitable ξ):

$$\begin{aligned}(d \circ \psi^{-1})(\xi) &= (g \circ \sigma_q \circ M \circ \psi^{-1})(\xi) \\ &= g \circ \sigma(q + MA^{-1}(\xi - b)) \\ &= g \circ \sigma(q - MA^{-1}b + MA^{-1}\xi) \\ &= (g \circ \sigma_{q-MA^{-1}b} \circ M \circ A^{-1})(\xi)\end{aligned} \tag{3}$$

Now, using (2) and (3), the expression in (1) transforms into:

$$\mathcal{A}' = g \circ \sigma_{q-MA^{-1}b} \circ M \circ A^{-1} \circ \sigma_{\tilde{v}} \circ \tilde{W}^{\dagger} \circ \sigma_{\tilde{h}+\tilde{W}b} \circ \tilde{W} \circ A \circ c.$$

Here, we used different colors to emphasize the alternation of connections and neuron layers. Thus, the whole unfolding and gluing operation results in a new autoencoder with two additional neuron layers. Now, the whole autoencoder can be again tuned with backpropagation. The unfolding operation can be repeated several times.

This procedure yields a whole stream of fine-tuned autoencoders of increasing depth and decreasing dimension of the innermost layer. The stream of autoencoders in some sense replaces an expensive metaparameter optimization: since we get multiple autoencoders, we can try out multiple compression dimensions, without having to rebuild the whole autoencoder from scratch each time.

The reason for using this training strategy in contrast to the one original suggested by Hinton *et al.* [3] is given in Figure (5). We evaluated the reconstruction error using two measures:

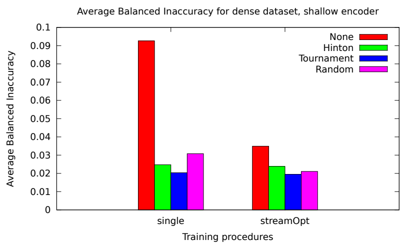
$$\text{Hamming Loss} = \frac{|Errors|}{|Entries|} \tag{4}$$

$$\text{Average Balanced Inaccuracy} = 1 - \frac{1}{2}\left(\frac{TP}{P} + \frac{TN}{N}\right) \tag{5}$$

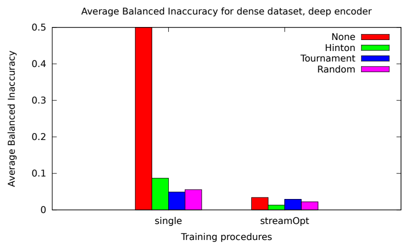
Where TP is the number of true positives, TN the number of true negatives, P the number of all positives, and N the number of all negatives.

We evaluated multiple training strategies for the autoencoders, and the results showed that in contrast to the strategy originally suggested by Hinton and Salakhutdinov [3], when using a stream of autoencoders, better results can be achieved. Additionally, there seems to be no big difference from using the parameters suggested by Hinton compared to using an optimization of these parameters.

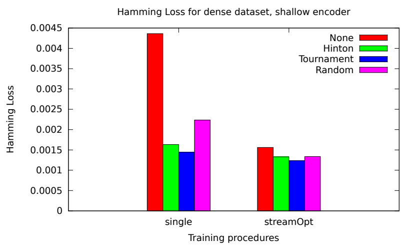
The next step is then to train a base classifier (a multi-target model) using the compressed labels as new target variables. In the previous step, we extracted the dependencies from the labels into the autoencoders, hence, in the best case,



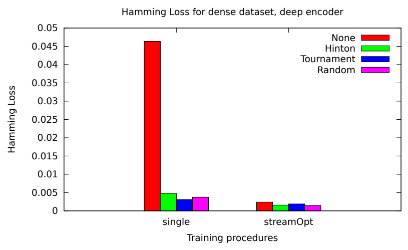
(a) 5 Layers – Balanced Inaccuracy



(b) 9 Layers – Balanced Inaccuracy



(c) 5 Layers – Hamming Loss



(d) 9 Layers – Hamming Loss

Fig. 5: Hamming Loss and balanced inaccuracy for multiple autoencoder training strategies on the *yeast* data set for different number of layers. `None` indicates no pretraining, `Hinton` uses the setup suggested by Hinton *et al.* [3], `Tournament` is the strategy used in this paper, where different setups are compared to each other and the best setup is used in the end. `Random` indicates a random selection. `single` is the strategy normally used in autoencoders and a reimplementation close to the approach in the implementation of Hinton, `streamOpt` is the stream of autoencoders described here, where multiple backpropagations are carried out. We trained the autoencoders on the data sets and then compressed and uncompressed the data sets to compare the reconstruction with the original data. In terms of computation, the running times of the `single` and `streamOpt` approaches are comparable (numbers not shown).

there are no dependencies left among the latent labels. Therefore, it should not be beneficial to use a sophisticated multi-target learner over a simple BR model. Nevertheless, if the autoencoder does not manage to extract all dependencies, it might be beneficial to use a more advanced multi-target learner. The training phase of the algorithm is put together by these two steps, and the model consists of the autoencoders and the multi-target (BR) model. The training phase is summarized in Algorithm (1) and shown in the Figure (1).

Algorithm 1: Pseudocode for training the classifier

```
Input: Data set with features $\mathcal{X}$ and labels $\mathcal{Y}$
Output: Autoencoder $\mathcal{A}$, threshold $t$, base learners $(f_i)_i$
$(\mathcal{X}_{train}, \mathcal{X}_{valid})$ := split($\mathcal{X}$)
$(\mathcal{Y}_{train}, \mathcal{Y}_{valid})$ := split($\mathcal{Y}$)
// Find the optimal compression dimension
foreach $\mathcal{A}$ in train_autoencoder_stream($\mathcal{Y}_{train}$) do
    $t$ := find_threshold($\mathcal{A}$,$\mathcal{Y}_{train}$)
    $\mathcal{Y}'$ := compress($\mathcal{A}$,$\mathcal{Y}_{train}$)
    foreach $i$ in $0 \dots q$ do
        // trains base learners on features $\mathcal{X}_{train}$ for
        // a single column $y'_i$ of target values
        $f_i$ := train($\mathcal{X}_{train}$, $y'_i$)
    end
    $p$ := predict($\mathcal{X}_{valid}$, $\mathcal{A}$, $t$, $(f_i)_i$)
    reconstructionError := distance($p$, $\mathcal{Y}_{valid}$)
end
// Now retrain with optimal compression dimension on
// the whole data set $(\mathcal{X}, \mathcal{Y})$ and return.
```

With an autoencoder, a threshold for binarization, and the trained base learners, one can easily predict labels for new instances (see Algorithm (2)). First, the multi-target model is applied and the latent labels are predicted. Next, the autoencoders are used to decompress the latent labels. The final labels are obtained by thresholding, so that the output is binary.

Algorithm 2: Pseudocode for predicting labels

```
Input: Features $\mathcal{X}$, autoencoder $\mathcal{A}$, threshold $t$, models $\{f_i\}_i$
Output: Predicted labels $\mathcal{Y}$
foreach $i$ in $0 \dots q$ do
    $y'_i$ := predict($f_i$, X)
end
$\mathcal{Y}_{real}$ := decompress($\mathcal{A}$,$\mathcal{Y}'$)
$\mathcal{Y}$ := binarize($\mathcal{Y}_{real}$, t)
```

The predicted bipartition is calculated from the confidences given from the autoencoder similar to other multi-label classifiers using a threshold. It should be noted that the calculated confidences are mostly close to 0 and 1, and rarely in between.

2 Experimental Results

Table 1: Data sets with their statistics that were used in the evaluation. Note that the table is sorted using the number of labels, the data set with the most labels first. Cardinality is the average number of positive labels per instance, density the fraction of positive labels in the whole data set, and distinct the number of distinct label combinations.

name	instances	nominal features	numeric features	labels	cardinality	density	distinct
CAL500 [4]	502	0	68	174	26.044	0.150	502
enron [5]	1702	1001	0	53	3.378	0.064	753
medical [6]	978	1449	0	45	1.245	0.028	94
genbase [7]	662	1186	0	27	1.252	0.046	32
birds [8]	645	2	258	19	1.014	0.053	133
yeast [9]	2417	0	103	14	4.237	0.303	198
flags [10]	194	9	10	7	3.392	0.485	54
scene [11]	2407	0	294	6	1.074	0.179	15
emotions [12]	593	0	72	6	1.869	0.311	27

Table 2: Evaluation using bipartition-based measures. Results for MANIAC are given both with BR and ECC as base classifier. The significant improvement ● or degradation ○ of MANIAC using BR as base classifier compared to the associated classifier is given. Note that the data sets are sorted according to the number of labels in descending order.

	Data set	MANIAC (BR)	BR	ECC	MANIAC (ECC)	MLC-BMaD	MANIAC (single layer)
Example-Based Accuracy	CAL500	0.25	0.21 ●	0.20 ●	0.25	0.01 ●	0.25
	enron	0.44	0.41 ●	0.42 ●	0.41 ●	0.32 ●	0.39 ●
	medical	0.54	0.46 ●	0.44 ●	0.47 ●	0.41 ●	0.17 ●
	genbase	0.74	0.93 ○	0.95 ○	0.65	0.90 ○	0.52 ●
	birds	0.57	0.59	0.55	0.60	0.57	0.53
	yeast	0.51	0.50 ●	0.51	0.52	0.50 ●	0.52
	flags	0.59	0.61 ○	0.62 ○	0.60	0.55 ●	0.61
	scene	0.58	0.60	0.58	0.56	0.47 ●	0.60
	emotions	0.52	0.53	0.55	0.53	0.48 ●	0.54 ○
	[○/ /●]		[2/3/4]	[2/4/3]	[0/7/2]	[1/1/7]	[1/5/3]
Example-Based FMeasure	CAL500	0.39	0.34 ●	0.33 ●	0.39	0.01 ●	0.39
	enron	0.56	0.52 ●	0.53 ●	0.53 ●	0.39 ●	0.50 ●
	medical	0.58	0.48 ●	0.46 ●	0.50 ●	0.43 ●	0.18 ●
	genbase	0.76	0.94 ○	0.95 ○	0.66	0.91 ○	0.54 ●
	birds	0.60	0.61	0.57	0.62	0.59	0.55 ●
	yeast	0.63	0.61 ●	0.62 ●	0.64	0.61 ●	0.63
	flags	0.72	0.73	0.73	0.72	0.66 ●	0.73
	scene	0.61	0.60	0.59	0.59	0.48 ●	0.61
	emotions	0.61	0.61	0.62	0.62	0.54 ●	0.64 ○
	[○/ /●]		[1/4/4]	[1/4/4]	[0/7/2]	[1/1/7]	[1/4/4]
Micro-Averaged FMeasure	CAL500	0.40	0.35 ●	0.33 ●	0.40	0.02 ●	0.39
	enron	0.58	0.55 ●	0.55 ●	0.57	0.44 ●	0.55 ●
	medical	0.65	0.60 ●	0.58 ●	0.59 ●	0.55 ●	0.27 ●
	genbase	0.82	0.96 ○	0.96 ○	0.79	0.93 ○	0.63 ●
	birds	0.43	0.40	0.31 ●	0.41	0.39 ●	0.28 ●
	yeast	0.65	0.64	0.64	0.65	0.64 ●	0.65
	flags	0.73	0.75 ○	0.75 ○	0.74	0.70	0.75
	scene	0.68	0.71 ○	0.70 ○	0.62 ●	0.62 ●	0.70
	emotions	0.65	0.67	0.68 ○	0.67	0.63 ●	0.67 ○
	[○/ /●]		[3/3/3]	[4/1/4]	[0/7/2]	[1/0/8]	[1/4/4]
Macro-Averaged FMeasure	CAL500	0.14	0.09 ●	0.07 ●	0.13	0.02 ●	0.08 ●
	enron	0.20	0.21	0.18 ●	0.19	0.18	0.14 ●
	medical	0.43	0.38 ●	0.36 ●	0.39	0.37 ●	0.29 ●
	genbase	0.68	0.82 ○	0.81 ○	0.66	0.78 ○	0.48 ●
	birds	0.27	0.22 ●	0.14 ●	0.24	0.22 ●	0.13 ●
	yeast	0.38	0.35 ●	0.35 ●	0.38	0.35 ●	0.37
	flags	0.61	0.65 ○	0.64	0.62	0.58	0.63
	scene	0.67	0.71 ○	0.70 ○	0.65	0.57 ●	0.69
	emotions	0.62	0.64	0.65 ○	0.64 ○	0.60 ●	0.64 ○
	[○/ /●]		[3/2/4]	[3/1/5]	[1/8/0]	[1/2/6]	[1/3/5]

Table 3: Evaluation using confidence-based measures. Results for Maniac are given both with BR and ECC as base classifier. The significant improvement • or degradation ◦ of Maniac using BR as base classifier compared to the associated classifier is given. Note that for *coverage*, *log loss*, and *one error*, an improvement in performance means a decrease in the measure. Hence the lower the value, the better the performance. Note that the data sets are sorted according to the number of labels.

	Data set	Maniac (BR)	BR	ECC	Maniac (ECC)	MLC -BMaD	Maniac (single layer)
Micro-Averaged AUC	CAL500	0.76	0.81 ◦	0.82 ◦	0.76	0.79 ◦	0.80 ◦
	enron	0.87	0.91 ◦	0.92 ◦	0.91 ◦	0.88	0.90 ◦
	medical	0.94	0.98 ◦	0.98 ◦	0.95	0.97 ◦	0.96 ◦
	genbase	0.98	1.00 ◦	1.00 ◦	0.97	0.99 ◦	0.97
	birds	0.83	0.92 ◦	0.92 ◦	0.85	0.91 ◦	0.89 ◦
	yeast	0.82	0.85 ◦	0.85 ◦	0.83	0.85 ◦	0.83
	flags	0.80	0.82 ◦	0.83 ◦	0.80	0.82 ◦	0.81
	scene	0.91	0.96 ◦	0.96 ◦	0.90	0.84 •	0.93 ◦
	emotions	0.83	0.87 ◦	0.87 ◦	0.84	0.86 ◦	0.85
	[◦/ /•]		[9/0/0]	[9/0/0]	[1/8/0]	[7/1/1]	[5/4/0]
Coverage	CAL500	151.91	134.61 ◦	129.91 ◦	149.73 ◦	146.79 ◦	141.10 ◦
	enron	25.83	12.94 ◦	11.77 ◦	27.07 •	12.94 ◦	30.49 ◦
	medical	6.58	1.70 ◦	1.66 ◦	7.86 •	1.86 ◦	14.56 ◦
	genbase	2.34	0.36 ◦	0.36 ◦	2.89	0.55 ◦	5.51 •
	birds	4.01	1.99 ◦	1.99 ◦	4.07	2.15 ◦	5.07 •
	yeast	7.19	6.19 ◦	6.08 ◦	7.00	6.21 ◦	7.03
	flags	3.71	3.66	3.62	3.70	3.61	3.73
	scene	0.66	0.37 ◦	0.37 ◦	0.79 •	1.08 •	0.74 •
	emotions	2.07	1.70 ◦	1.69 ◦	1.98	1.73 ◦	2.02
	[◦/ /•]		[8/1/0]	[8/1/0]	[1/5/3]	[1/1/7]	[3/3/3]
Log Loss	CAL500	183.14	64.75 ◦	58.98 ◦	175.00	117.18 ◦	150.65 ◦
	enron	33.67	8.77 ◦	7.35 ◦	33.20	9.72 ◦	37.09 ◦
	medical	9.72	2.39 ◦	2.33 ◦	11.16 •	2.56 ◦	17.52 ◦
	genbase	4.38	1.00 ◦	0.98 ◦	6.13 •	1.18 ◦	9.55 •
	birds	10.78	2.57 ◦	2.39 ◦	11.01	2.83 ◦	14.71 •
	yeast	25.23	6.07 ◦	6.02 ◦	23.77	6.34 ◦	26.11
	flags	12.08	3.74 ◦	3.59 ◦	14.27	5.26 ◦	14.04
	scene	7.09	1.32 ◦	1.33 ◦	8.17	4.07 ◦	7.16
	emotions	13.33	2.49 ◦	2.47 ◦	11.91	2.83 ◦	11.25
	[◦/ /•]		[9/0/0]	[9/0/0]	[0/7/2]	[9/0/0]	[3/4/2]
One Error	CAL500	0.23	0.14 ◦	0.13 ◦	0.23	0.39 •	0.11 ◦
	enron	0.27	0.21 ◦	0.21 ◦	0.27	0.32 •	0.27
	medical	0.31	0.20 ◦	0.22 ◦	0.36 •	0.20 ◦	0.58 ◦
	genbase	0.12	0.00 ◦	0.00 ◦	0.11	0.00 ◦	0.21
	birds	0.73	0.64 ◦	0.65 ◦	0.73	0.64 ◦	0.76 •
	yeast	0.24	0.23	0.24	0.24	0.23 ◦	0.24
	flags	0.22	0.22	0.21	0.24	0.24	0.22
	scene	0.26	0.18 ◦	0.19 ◦	0.30 •	0.32 •	0.25
	emotions	0.31	0.26 ◦	0.25 ◦	0.29	0.28	0.28
	[◦/ /•]		[7/2/0]	[7/2/0]	[0/7/2]	[4/2/3]	[2/6/1]

References

1. J. Wicker, B. Pfahringer, and S. Kramer, "Multi-label classification using Boolean matrix decomposition," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing.* ACM, 2012, pp. 179–186.
2. F. Tai and H.-T. Lin, "Multilabel classification with principal label space transformation," *Neural Computation*, vol. 24, no. 9, pp. 2508–2542, 2012.

3. G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
4. D. Turnbull, L. Barrington, D. Torres, and G. Lanckriet, "Semantic annotation and retrieval of music and sound effects," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 16, no. 2, pp. 467–476, 2008.
5. B. Klimt and Y. Yang, "The enron corpus: A new dataset for email classification research," in *Machine learning: ECML 2004*. Springer, 2004, pp. 217–226.
6. J. P. Pestian, C. Brew, P. Matykiewicz, D. Hovermale, N. Johnson, K. B. Cohen, and W. Duch, "A shared task involving multi-label classification of clinical free text," in *Proceedings of the Workshop on BioNLP 2007: Biological, Translational, and Clinical Language Processing.* Association for Computational Linguistics, 2007, pp. 97–104.
7. S. Diplaris, G. Tsoumakas, P. A. Mitkas, and I. Vlahavas, "Protein classification with multiple algorithms," in *Advances in Informatics.* Springer, 2005, pp. 448–456.
8. F. Briggs, Y. Huang, R. Raich, K. Eftaxias, Z. Lei, W. Cukierski, S. F. Hadley, A. Hadley, M. Betts, X. Z. Fern *et al.*, "The 9th annual mlsp competition: New methods for acoustic classification of multiple simultaneous bird species in a noisy environment," in *Machine Learning for Signal Processing (MLSP), 2013 IEEE International Workshop on.* IEEE, 2013, pp. 1–8.
9. A. Elisseeff and J. Weston, "A kernel method for multi-labelled classification," in *Advances in neural information processing systems*, 2001, pp. 681–687.
10. E. C. Gonçalves, A. Plastino, and A. A. Freitas, "A genetic algorithm for optimizing the label ordering in multi-label classifier chains," in *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on.* IEEE, 2013, pp. 469–476.
11. M. R. Boutell, J. Luo, X. Shen, and C. M. Brown, "Learning multi-label scene classification," *Pattern recognition*, vol. 37, no. 9, pp. 1757–1771, 2004.
12. K. Trohidis, G. Tsoumakas, G. Kalliris, and I. P. Vlahavas, "Multi-label classification of music into emotions." in *ISMIR*, vol. 8, 2008, pp. 325–330.