



2011-10-11

Improving Multi-label Classification by Avoiding Implicit Negativity with Incomplete Data

Derrall L. Heath

Brigham Young University - Provo

Follow this and additional works at: <https://scholarsarchive.byu.edu/etd>

 Part of the [Computer Sciences Commons](#)

BYU ScholarsArchive Citation

Heath, Derrall L., "Improving Multi-label Classification by Avoiding Implicit Negativity with Incomplete Data" (2011). *All Theses and Dissertations*. 2844.

<https://scholarsarchive.byu.edu/etd/2844>

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

Improving Multi-label Classification by Avoiding Implicit Negativity
with Incomplete Data

Derrall Heath

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of
Master of Science

Dan Ventura, Chair
Tony Martinez
Scott Woodfield

Department of Computer Science
Brigham Young University
December 2011

Copyright © 2011 Derrall Heath
All Rights Reserved

ABSTRACT

Improving Multi-label Classification by Avoiding Implicit Negativity with Incomplete Data

Derrall Heath

Department of Computer Science, BYU
Master of Science

Many real world problems require multi-label classification, in which each training instance is associated with a set of labels. There are many existing learning algorithms for multi-label classification; however, these algorithms assume implicit negativity, where missing labels in the training data are automatically assumed to be negative. Additionally, many of the existing algorithms do not handle incremental learning in which new labels could be encountered later in the learning process. A novel multi-label adaptation of the backpropagation algorithm is proposed that does not assume implicit negativity. In addition, this algorithm can, using a naïve Bayesian approach, infer missing labels in the training data. This algorithm can also be trained incrementally as it dynamically considers new labels. This solution is compared with existing multi-label algorithms using data sets from multiple domains and the performance is measured with standard multi-label evaluation metrics. It is shown that our algorithm improves classification performance for all metrics by an overall average of 7.4% when at least 40% of the labels are missing from the training data, and improves by 18.4% when at least 90% of the labels are missing.

Keywords: implicit negativity, multi-label classification, thesis

ACKNOWLEDGMENTS

I'd like to thank Dan Ventura, my fantastic advisor, who helped me to become a better writer and researcher; David Norton, a fellow lab member, who helped me polish many of my ideas and solutions; and Tony Martinez as a committee member. I'd also like to thank my family for their constant encouragement.

Contents

Contents	iv
1 Introduction	1
2 Related Work	4
2.1 Problem Transformational Methods	4
2.2 Algorithm Adaptation Methods	5
2.3 Implicit Negativity	6
2.4 Inferring Missing Labels	8
2.5 Incremental Learning	9
3 Methods	10
3.1 Avoiding Implicit Negativity	10
3.2 Inferring Missing Labels	12
3.3 Incremental Learning	14
4 Experimental Setup	16
4.1 Data Sets	16
4.2 Algorithms	18
4.3 Evaluation Metrics	18
5 Results	22
5.1 Artificially Removing Labels	22
5.2 The DARCI Data Set	27
5.3 Only One Positive Label	29

5.4 Incremental Learning	32
5.5 Comparing with the Bayesian Model	35
6 Conclusions and Future Work	37
References	39

Chapter 1

Introduction

Traditionally, many classification problems have dealt with single-label classification, meaning each training instance is associated with only one class or label. In single-label classification, the task is to learn some target function $f : X \rightarrow L$ that predicts the correct label for each new instance. However, in multi-label classification, each training instance can be associated with more than one label. The task is to learn some target function $f : X \rightarrow 2^L$ that predicts the correct set of labels (of unknown size) for each new instance.

There are many interesting problems that require multi-label classification. For example, in gene classification, genes can perform more than one function [1]. In text categorization, a document can contain multiple topics such as outdoors, sports, and recreation [2]. Images can be labeled by the multiple objects they may contain [3]. Web sites can be given several labels for the different topics they represent [4]. Music and movies can belong to more than one genre [5]. The list of multi-label classification problems continues to grow and it is imperative that we have good multi-label classification solutions for these various problems.

The existing methods for multi-label classification follow two main strategies: *problem transformational methods* and *algorithm adaptation methods* [6]. Problem transformational methods involve transforming a multi-label classification problem into one or more single-label classification problems. Algorithm adaptation methods involve modifying specific learning algorithms to directly handle multi-label problems. Much research has been done to show that many of these methods are successful in solving various multi-label problems.

However, all these multi-label learning algorithms assume that each training instance will have all the correct positive labels provided and that any label not listed is negative. We call this assumption *implicit negativity*. We will define our set of multi-label data to be $\{x_i, Y_i\}$, $i = 1 \dots m$, where $Y_i \subseteq L$ is the set of all correct positive labels for x_i and $L = \{\lambda_j : j = 1 \dots q\}$ is the set of all possible labels. Let Y'_i be the provided set of positive labels for a training instance x_i . Existing multi-label learning algorithms assume that $Y'_i = Y_i$, or that the provided set of positive labels is always equal to the true set of all positive labels for a training instance x_i . In reality, this is not always the case. Gathering training data is constantly an issue in machine learning and getting training data with all the correct positive labels listed can be very difficult in many domains. Often Y'_i is only a subset of Y_i and is therefore missing positive labels. Algorithms that assume implicit negativity struggle with these incomplete positive label sets because it is automatically implied that those missing positive labels are negative and the model is trained incorrectly. It should be noted that there is also the case involving noise, where $\lambda_j \in Y'_i$ but $\lambda_j \notin Y_i$. However, dealing with noise is beyond the scope of this thesis.

Removing the assumption of implicit negativity helps to avoid training the model incorrectly when positive labels are missing. However, eliminating this assumption requires negative labels, in addition to the positive labels. This can potentially create a class imbalance problem between the positive and negative examples of each label because negative labels are generally even harder to acquire than positive labels. One approach to dealing with this is to have the learning algorithm take advantage of label correlations in the data to infer what the missing positive and negative labels are for each training instance. Let \bar{Y}_i be the true set of all negative labels for a training instance x_i and let \bar{Y}'_i be the provided set of negative labels for a training instance x_i . When the learning algorithm encounters a new training instance x_i with positive labels Y'_i and negative labels \bar{Y}'_i , it should be able to infer Y_i and \bar{Y}_i .

Another issue to consider in multi-label learning is that during incremental training, often the complete set of all possible labels L may not be known *a priori*. In these situations,

the learning algorithm must be able to dynamically account for new labels as they are encountered. Let L' be the set of all possible labels that are known at the start of training. When the learning algorithm encounters a new label $\lambda \notin L'$, it should be able to dynamically add λ to L' and start learning this new label without disrupting what has previously been learned.

We propose a novel variation of the backpropagation algorithm called the BAIN (Backpropagation for Avoiding Implicit Negativity) algorithm that does not assume implicit negativity. The BAIN algorithm only trains the output nodes explicitly labeled (as positive or negative) in the training data, while ignoring the labels not mentioned. The BAIN algorithm can also dynamically add new output nodes for previously unseen labels during incremental training. Since the BAIN algorithm does not assume implicit negativity, it is now reliant on explicitly given negative and positive labels. The BAIN algorithm uses a naïve Bayes method to infer missing positive and negative labels in the training data.

BAIN is compared to popular multi-label learning algorithms, including BP-MLL [1], Binary Relevance [6] and ML-kNN [7]. Four experiments with two different types of data sets are outlined. Standard multi-label evaluation metrics such as *Hamming loss*, *accuracy*, *precision*, *recall*, *one-error*, *coverage*, and *ranking loss* [2, 1] are used to evaluate the effectiveness of our new BAIN algorithm compared to existing multi-label algorithms. It is shown that the BAIN algorithm is robust to missing labels in the training data and outperforms existing multi-label learning methods as the amount of missing data increases.

Chapter 2

Related Work

Problem transformational methods and algorithm adaptation methods are the two main strategies for handling multi-label classification. Here we review a few of the most popular methods. For a more comprehensive review of problem transformational and algorithm adaptation methods, there are several excellent surveys on multi-label classification [8, 6, 9]. In addition, we will consider other work that has been done to solve the specific problems of implicit negativity, inferring missing labels, and incremental learning.

2.1 Problem Transformational Methods

Problem transformational methods involve transforming a multi-label classification problem into one or more single-label classification problems. One approach is to treat each unique set of labels in the training data as an additional label in the set of all possible labels L [6, 9]. For example, given a data set with the possible labels $L = \{A, B, C, D\}$ and training set:

$$\begin{aligned}x_1 &\rightarrow A \\x_2 &\rightarrow A, B \\x_3 &\rightarrow A, C, D \\x_4 &\rightarrow D\end{aligned}$$

The set of possible labels becomes $L = \{A, B, C, D, \{A, B\}, \{A, C, D\}\}$ and the problem is now a single-label classification problem and any standard learning algorithm can be used. This method does not work well when there are a large number of possible labels because it is difficult to get enough data to support every combination of labels that might be encountered.

Another strategy is to split this problem into several binary classification problems, one for each label [6, 9]. So the data set mentioned previously becomes four new data sets:

$$\begin{array}{llll}
 L_A = \{A, \bar{A}\} & L_C = \{C, \bar{C}\} & L_B = \{B, \bar{B}\} & L_D = \{D, \bar{D}\} \\
 x_1 \rightarrow A & x_1 \rightarrow \bar{C} & x_1 \rightarrow \bar{B} & x_1 \rightarrow \bar{D} \\
 x_2 \rightarrow A & x_2 \rightarrow \bar{C} & x_2 \rightarrow B & x_2 \rightarrow \bar{D} \\
 x_3 \rightarrow A & x_3 \rightarrow C & x_3 \rightarrow \bar{B} & x_3 \rightarrow D \\
 x_4 \rightarrow \bar{A} & x_4 \rightarrow \bar{C} & x_4 \rightarrow \bar{B} & x_4 \rightarrow D
 \end{array}$$

This method is called Binary Relevance and any standard binary classifier can now be used for each label. A common criticism of this method is that, rather than a single model, multiple learning models are needed. This can be inefficient for problems with large label sets. Another common criticism is that as labels are separated, correlations between labels are not considered which can weaken the system's expressive power [1].

2.2 Algorithm Adaptation Methods

Algorithm adaptation methods involve modifying specific learning algorithms to directly handle multi-label problems. One of the first algorithms to be adapted to multi-label classification was the C4.5 algorithm [10]. The C4.5 algorithm was modified to allow multiple labels in the leaves of the tree and the entropy formula was changed to consider both the class membership and non-class membership of each label. The AdaBoost algorithm was extended for multi-label classification resulting in the Adaboost.MH and the Adaboost.MR algorithms [2]. Adaboost.MH focuses on label classification, while Adaboost.MR focuses on label ranking.

ML-kNN is a multi-label classification algorithm adapted from the kNN algorithm [7]. This adaptation uses the kNN algorithm independently for each label; so fundamentally it is a problem transformational method. However, it differs from a normal problem transformational method because it makes use of prior and posterior probabilities as it recombines the results. There also exists an SVM algorithm adapted for multi-label classification that is also in

reality a problem transformational method [11]. This SVM adaptation does, however, use a kind of meta-learning strategy to consider the dependencies among the different labels.

BP-MLL (Backpropagation for Multi-Label Learning) is a modified version of the backpropagation algorithm [1]. The error function is modified to consider label correlations where labels belonging to an instance should be ranked higher than those not belonging to that instance. ML-RBF is an extension of the RBF neural network algorithm that handles multi-label classification [12]. ML-RBF selects hidden nodes by conducting a clustering analysis on instances of each possible label. Information encoded in the hidden nodes corresponding to all classes is exploited to optimize the weights corresponding to each label.

Multi-label classification can be thought of as a specific case of structured output learning. Structured output learning is a classification task for in which the output space consists of structured objects, such as trees, strings, sequences, or graphs. The goal is to learn the entire structure of the output, where multiple instances are inter-related. There are several structured learning algorithms that have been used for multi-label classification [13, 14, 15]. These methods combine the advantages of kernel-based and probabilistic classifiers. The kernel-based component can deal with high dimensional feature spaces and provides strong generalization guarantees, while the probabilistic component is able to represent label correlations and exploit problem structure. A more comprehensive review of structured output learning can be found here [16].

2.3 Implicit Negativity

Most of these multi-label classification methods work well for the specific tasks they were developed for. However, each of these methods assume implicit negativity, and will likely perform poorly when applied to tasks where not all the correct positive labels are provided.

The Weakened Implicit Negatives (WIN) algorithm was proposed in order to deal with the issue of implicit negativity for backpropagation [17]. The WIN algorithm uses a separate probabilistic neural network component that learns a target output value for each output

node in the network. When a label is missing for a given training instance, the WIN algorithm uses the learned target output value during error calculation instead of assuming it to be zero. Training time is alternated between learning the network weights and learning the target output values. How often each component alternates must be provided as a parameter. This method was shown to be effective at learning toy problems and a few real world problems with incomplete data; however, the algorithm was not compared to any of the standard multi-label algorithms nor with any of the standard multi-label evaluation metrics. Additionally, using a separate probabilistic neural network on top of backpropagation increases the computational complexity, which could be problematic for problems with large numbers of labels. Finally, the WIN algorithm was designed for the specific case where $|Y_i'| = 1$ and $|\bar{Y}_i'| = 0$. In other words, there is exactly one positive label provided for each training instance and all other labels are unknown.

In the broader field of structured output learning, there exists several methods that deal with missing labels for sequence labeling and part of speech tagging [18, 19, 20]. Each method uses a probabilistic model for which the parameters must be optimized to match the training data. They deal with missing labels by marginalizing out the the unknown labels so as to maximize the likelihood of a set of possible label structures which are consistent with the given data. These methods are only applied to sequence labeling and part of speech tagging and it is unclear how they would adapt to multi-label classification.

However, a multi-label active learning algorithm was proposed that is based on similar probabilistic ideas [21]. This method avoids the tedious process of labeling thousands of images with possibly hundreds of labels by using a Bayesian error bound to actively annotate only a subset of labels and then later deal with the missing labels. This approach then uses the Maximum Entropy Method to learn a function that can classify new instances [22]. This method handles missing labels by integrating out the unlabeled part yielding the marginal distribution of the labeled part. This technique was shown to be effective with small data

sets. However, this algorithm is computationally inefficient for problems with large sets of possible labels.

2.4 Inferring Missing Labels

Removing the assumption of implicit negativity helps to avoid training the model incorrectly when positive labels are missing. However, to compensate, negative labels are needed in addition to the positive labels. Inferring those missing positive and negative labels is needed to help fill in the gaps in the training data. That is to say, when the learning algorithm encounters a new training instance x_i with a set of incomplete positive labels Y'_i and negative labels \bar{Y}'_i , it should be able to infer the true set of positive labels Y_i and negative labels \bar{Y}_i . This is similar to the class imbalance problem because there will likely be fewer negative examples of each label compared to positive examples of each label.

A common solution to class imbalance is to either oversample the minority class or under-sample the majority class [23]. This solution is not the most effective because over-fitting becomes a problem with oversampling and loss of crucial data is a problem with under-sampling. Another method is to use data generation strategies where new data instances for the minority class are generated by interpolating between the existing data instances [24, 25]. In multi-label classification, class imbalance has been studied [26]. However, the research does not deal with class imbalance between positive and negative examples caused by missing data. Rather, the research deals with class imbalance between different labels when one label is more rare than another.

There is little research that deals with inferring missing labels for multi-label problems. The active annotation algorithm mentioned in the previous section indirectly infers missing labels as part of the training process [21]. However, there is no algorithm that we are aware of that explicitly tries to predict missing labels in the training data by considering label correlations in the data that is provided.

2.5 Incremental Learning

Most of these multi-label learning algorithms are not incremental learners and specifically do not deal with the case of the complete set of all possible labels being unknown *a priori*. There are learning algorithms that deal with incremental learning for neural networks and handle the issue of new incoming labels [27, 28]. However, these incremental algorithms are not designed for multi-label classification. There is an active annotation algorithm that claims to be the first incremental multi-label algorithm [29, 21]. This algorithm allows for introducing new labels and is able to dynamically update the model to account for these new incoming labels. This solution is similar to our approach, except our solution is a backpropagation solution, while theirs is a Bayesian model.

Chapter 3

Methods

Multi-label learning is a complicated problem with many potential issues. This thesis proposes a novel variation of the backpropagation algorithm called the BAIN (Backpropagation for Avoiding Implicit Negativity) algorithm that addresses multi-label learning and deals with the issues previously mentioned. Specifically, the BAIN algorithm:

- Does not assume Implicit Negativity
- Uses label correlations in the training data to infer missing labels
- Can learn incrementally and dynamically incorporate previously unseen labels

These issues are nontrivial when it is difficult to get data with all the correct labels for each training instance or when new data is being gathered and additional labels could appear. The BAIN algorithm is implemented by taking the initial set of training data and building a standard feed-forward neural network with each output node representing a label. The neural network is then trained using standard backpropagation with a few key differences.

3.1 Avoiding Implicit Negativity

When certain labels are encountered in the training data, only the weights of the outputs nodes corresponding to those labels are trained, while the other output nodes are ignored. The assumption of implicit negativity is simply not made. Negative examples must be explicitly labeled as negative in the training data. For example, given a data set with the possible labels $L = \{A, B, C, D\}$ and training set:

$$\begin{aligned}
x_1 &\rightarrow A, \bar{C} \\
x_2 &\rightarrow A, B \\
x_3 &\rightarrow A, \bar{B}, C, D \\
x_4 &\rightarrow \bar{B}, D
\end{aligned}$$

When training the neural network on x_1 , only the weights corresponding to output nodes A and C are trained, while B and D are ignored. For x_2 , only the weights corresponding to output nodes A and B are trained, while C and D are ignored.

This is done by changing how the error term is calculated for each network output node. Let o_k , t_k , and δ_k be the actual output value, target output value, and error term respectively for the k th output node. In standard backpropagation, the error term δ_k is calculated for each output node k as follows:

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

The error term δ_h for each hidden node h can then be calculated:

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in \text{outputs}} w_{kh} \delta_k$$

Where o_h is the output value for hidden node h and w_{kh} is the weight between hidden node h and output node k . The network weights are then updated as follows:

$$w_{ji} \leftarrow w_{ji} + \eta \delta_j x_{ji}$$

Where η is the learning rate, x_{ji} is the input value to node j from unit i , and w_{ji} denotes the corresponding weight.

In the BAIN algorithm, the assumption of implicit negativity is removed by changing how the error term δ_k is calculated:

$$\delta_k \leftarrow \begin{cases} 0, & \text{if } t_k \text{ is unknown} \\ o_k(1 - o_k)(t_k - o_k), & \text{otherwise} \end{cases}$$

When using standard backpropagation for multi-label classification, t_k is automatically assumed to be zero when its value is unknown. However, by setting δ_k to zero when t_k is unknown, the weights that connect each hidden node to the k th output node are prevented from changing due to multiplication by zero. The k th output node is also prevented from affecting the error term for each hidden node. In this way the k th output node is ignored when there is no explicitly given label for that node.

3.2 Inferring Missing Labels

The goal is to take advantage of label correlations in the data to infer the missing positive and negative labels for each training instance. To do this, a naïve Bayes approach is used to infer the missing labels. Naïve Bayes is simple and fast, which may be needed to avoid long computations that could take longer than training the actual multi-label model. The naïve Bayes classifier takes the form:

$$\text{classify}(w_1, w_2, \dots, w_n) = \underset{c}{\operatorname{argmax}} p(C = c) \prod_{j=1}^n p(W_j = w_j | C = c)$$

In our case, C is the missing label we are trying to infer, which can be either positive or negative. w_j is the set of labels $\lambda \in Y_i' \cup \bar{Y}_i'$ or all the provided labels (both positive and negative) for a given training instance x_i . The missing label is inferred only when there is confidence in the prediction. This means that the argmax is only taken when the difference is larger than a certain threshold θ , otherwise the label is left unknown. Hence, our method

to infer a missing label C can be defined as follows:

$$C = \begin{cases} 1, & \text{if } f(1, Y'_i, \bar{Y}'_i) - f(-1, Y'_i, \bar{Y}'_i) > \theta \\ -1, & \text{if } f(-1, Y'_i, \bar{Y}'_i) - f(1, Y'_i, \bar{Y}'_i) > \theta \\ \text{unknown}, & \text{otherwise} \end{cases}$$

where

$$f(c, Y'_i, \bar{Y}'_i) = p(C = c) \prod_{\lambda \in Y'_i} p(\lambda | C = c) \prod_{\lambda \in \bar{Y}'_i} p(\bar{\lambda} | C = c)$$

This method is applied to each missing label C in each training instance x_i . The probabilities can be calculated from the training data based on the assumption that there exist other training instances that are not missing the label C .

For example, given a data set with the possible labels $L = \{A, B, C, D\}$ and training set:

$$\begin{aligned} x_1 &\rightarrow A, B, \bar{C}, \bar{D} \\ x_2 &\rightarrow \bar{A}, \bar{B}, C, D \\ x_3 &\rightarrow A, \bar{C}, \bar{D} \\ x_4 &\rightarrow \bar{A}, \bar{B}, C, D \\ x_5 &\rightarrow A, B, C, \bar{D} \end{aligned}$$

Information about B is missing from x_3 and must be inferred. For training instance x_3 , $Y'_3 = \{A\}$ and $\bar{Y}'_3 = \{C, D\}$, hence $f(1, Y'_3, \bar{Y}'_3)$ and $f(-1, Y'_3, \bar{Y}'_3)$ can be calculated as follows:

$$f(1, Y'_3, \bar{Y}'_3) = p(B)p(A|B)p(\bar{C}|B)p(\bar{D}|B)$$

$$f(-1, Y'_3, \bar{Y}'_3) = p(\bar{B})p(A|\bar{B})p(\bar{C}|\bar{B})p(\bar{D}|\bar{B})$$

The probability terms on the right hand side of each equation are calculated by counting the occurrences of each label in the training set. For instance, $p(B) = 0.5$ because positive B occurs twice out of the four total times B occurs. $p(A|B) = 1.0$ because every time B is positive, so is A . By counting occurrences in this manner, the rest of the probability terms

```

if  $\lambda_j \notin L'$  then
   $L' \leftarrow L' \cup \lambda_j$ 
   $o_j = createOutputNode(\lambda_j)$ 
   $outputNodes \leftarrow outputNodes \cup o_j$ 
  for all  $h \in hiddenNodes$  do
     $addWeight(h, o_j)$ 
  end for
end if

```

Figure 3.1: Psuedocode for dynamically incorporating a new label into the neural network during incremental training.

are calculated: $p(\bar{B}) = 0.5$, $p(\bar{C}|B) = 0.5$, $p(\bar{D}|B) = 1.0$, $p(A|\bar{B}) = 0.0$, $p(\bar{C}|\bar{B}) = 0.0$, and $p(\bar{D}|\bar{B}) = 0.0$. The equations can then be evaluated as follows:

$$f(1, Y'_3, \bar{Y}'_3) = 0.5 * 1.0 * 0.5 * 1.0 = 0.25$$

$$f(-1, Y'_3, \bar{Y}'_3) = 0.5 * 0.0 * 0.0 * 0.0 = 0.0$$

If $(0.25 - 0.0) > \theta$, then it can be safely inferred that the missing label B is positive.

3.3 Incremental Learning

In incremental learning, all the possible labels that might be encountered during training are not always known *a priori*. Recall that L' is the set of all possible labels that are known at the start of training. If a previously unknown label λ is encountered where $\lambda \notin L'$, then there would be no output node in the network that corresponds to that label. To accommodate a new incoming label, a new output node is simply added to the output layer that corresponds to that label. The weights to the hidden layer are initialized randomly and the model can now begin learning this new label. Pseudocode for accommodating new incoming labels is shown in Figure 3.1.

As an example, if the initial training set had the possible labels $L' = \{A, B, C, D\}$, then the network would have four output nodes, one corresponding to each label. If a new

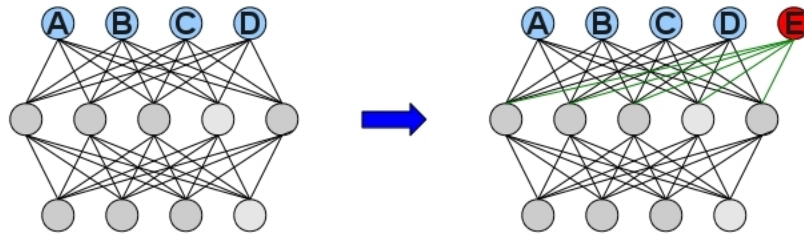


Figure 3.2: Dynamically adding a new label to the neural network during incremental training.

label E is encountered during incremental training, E would be added to L' and a fifth output node would be dynamically created in the network with randomly initialized weights as shown in Figure 3.2. The BAIN algorithm can now train the network as if that output node had been known from the start. In fact, the resulting solution will be identical to the solution obtained if label E was known from the beginning. This is because no assumption of implicit negativity is made and that output node would have been ignored if the training instances did not explicitly mention E .

Chapter 4

Experimental Setup

Four experiments are designed to validate our proposed solution. The first experiment involves several well known multi-label data sets that have been used in other studies and are publicly available [30]. These data sets do not have missing labels; however, this enables us to artificially remove both positive and negative labels from the data sets and then compare the results with the complete data sets. The percentage of labels that are removed can be adjusted, which allows us to evaluate how the algorithm performs with different numbers of missing labels. The second experiment involves the DARCI data set [31], which is a real world data set with a large percentage of missing labels. This second experiment shows how our solution performs with actual problems that have missing labels. The third experiment involves the specific case where there is only one positive label provided for each instance. This allows us to compare BAIN with the WIN algorithm [17], which is a multi-label learning algorithm that deals with that specific case. The fourth experiment tests how well our algorithm can learn incrementally. Using the data sets from the first set of experiments, we explicitly leave out labels and training instances from the training data and then later introduce them. The final result is then compared to a model that is trained with all the training data from the start.

4.1 Data Sets

Eight of the ten data sets used in the experiments are publicly available as part of the open source Mulan project [30]. These data sets represent a variety of domains and a

Table 4.1: Data sets that are used for experimentation.

Data Set	Domain	Instances	Nominal Attributes	Numeric Attributes	Labels	Cardinality	Density
corel5k	text	5000	499	0	374	2.028	0.009
medical	text	978	1449	0	45	1.245	0.028
emotions	music	593	0	72	6	1.859	0.311
eron	text	1702	1001	0	53	3.378	0.064
genbase	biology	662	1186	0	27	1.252	0.046
mediamill	video	43907	0	120	101	4.376	0.043
scene	images	2407	0	294	6	1.074	0.179
yeast	biology	2417	0	103	14	4.237	0.303
triclass	artificial	1500	0	2	8	3.00	0.375
classoverlap	artificial	1500	0	2	8	1.578	0.197
DARCI	images	2101	0	102	211	3.535	0.017
Average		5706.09	375.91	63.18	77.55	2.506	0.143

range of statistical properties. The remaining two data sets (triclass and classoverlap) are simple artificially generated toy problems [17]. The attributes of each data set can be seen in Table 4.1. The table introduces two potentially unfamiliar terms used for quantifying properties of multi-label data sets. *Cardinality*, or label cardinality, refers to the average number of labels per instance. *Density*, or label density, is equivalent to the label cardinality divided by the number of possible labels. This metric gives an indication of how often labels are used throughout the data set.

The DARCI data set consists of 2,101 images that are labeled with adjectives that describe each image. The DARCI data set has 102 numerical features and currently 211 possible labels with a cardinality of 3.535 and density of 0.017. Each image has, on average, only 4.7% of both positive and negative labels; the rest are unknown. Additionally, data is continually being collected, and new labels (or adjectives) are continually being added. This makes the DARCI data set ideal for testing our solution as it is a real world example of the problems we are trying to solve.

4.2 Algorithms

For a baseline comparison in the first three experiments, popular multi-label algorithms which include BP-MLL [1], Binary Relevance [6] and ML-kNN [7] are used. Two versions of the BAIN algorithm are also compared. The first is the full algorithm as described in Chapter 3. The second is a version that does not infer missing labels and is referred to as BAIN_nopred. This allows us to evaluate how effective our method is at inferring missing labels, as opposed to only removing the assumption of implicit negativity. Additionally, an untrained model that is random in its label predictions will be used to provide an overall baseline for comparison.

The number of k -nearest neighbors for the ML-kNN algorithm is the same for each data set at $k = 5$. The binary relevance method uses standard single-label backpropagation as its base algorithm. The various parameters used by each neural network based algorithm for each data set can be seen in Table 4.2. These parameters are not necessarily optimal for each algorithm; reasonable parameters were chosen based on simple trial runs with each data set. The neural network parameters were calibrated with the BPMLL algorithm and then used for all neural network based algorithms.

4.3 Evaluation Metrics

Empirically evaluating multi-label problems is more complicated than evaluating single-label problems, as there are different degrees of correctness. Multi-label evaluation metrics fall into two main categories: *prediction-based* and *ranking-based*. Prediction-based metrics evaluate how well the algorithm predicts the actual set of correct labels for each instance. Ranking-based metrics evaluate how well the algorithm ranks the labels relative to one another. The correct labels should be ranked higher than the incorrect labels. The same notation as established in Chapter 1 will be used to formalize our evaluation metrics. In addition, given an instance x_i , the set of predicted labels is denoted as Z_i , while the predicted rank of a

Table 4.2: Parameters used for the neural network based algorithms BAIN, BPMLL, WIN, and binary relevance. The number of hidden nodes for binary relevance are shown as hidden nodes per model, where the number of models is equal to the number of possible labels in the data set.

Data Set	Epochs	Learning Rate	Number of Hidden Nodes	
			BAIN, WIN, BP-MLL	Binary Relevance
corel5k	100	.01	256	2
medical	100	.01	128	10
emotions	100	.01	96	10
eron	100	.01	128	10
genbase	100	.01	256	10
mediamill	100	.01	128	10
scene	100	.01	96	10
yeast	100	.05	96	10
triclass	100	.10	16	2
classoverlap	100	.10	16	2
DARCI	200	.01	128	10

label λ is denoted as $r_i(\lambda)$. We will use the following standard multi-label prediction-based evaluation metrics [8] with 10-fold cross validation as we compare each algorithm:

Hamming Loss is the average percentage of correct labels not predicted and incorrect labels predicted.

$$HammingLoss = \frac{1}{m} \sum_{i=1}^m \frac{|Y_i \Delta Z_i|}{q}$$

where Δ is the symmetric difference between the two sets.

Accuracy is the average percentage of true positives out of the total true positives, false positives, and false negatives.

$$Accuracy = \frac{1}{m} \sum_{i=1}^m \frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|}$$

Precision is the average percentage of predicted labels that were correct.

$$Precision = \frac{1}{m} \sum_{i=1}^m \frac{|Y_i \cap Z_i|}{|Z_i|}$$

Recall is the average percentage of correct labels that were predicted.

$$Recall = \frac{1}{m} \sum_{i=1}^m \frac{|Y_i \cap Z_i|}{|Y_i|}$$

We will use the following standard multi-label ranking-based evaluation metrics [2, 1] with 10-fold cross validation as we compare each algorithm:

One-Error is the percentage of top ranked labels that are not in the set of correct labels.

$$OneError = \frac{1}{m} \sum_{i=1}^m \delta(\operatorname{argmin}_{\lambda \in L} r_i(\lambda))$$

where

$$\delta(\lambda) = \begin{cases} 1, & \text{if } \lambda \notin Y_i \\ 0, & \text{otherwise} \end{cases}$$

Coverage is how far, on average, we need to go down the list of predicted labels in order to cover all the correct labels. It is normalized between 0 and 1.

$$Coverage = \frac{1}{m} \sum_{i=1}^m \frac{\omega - |Y_i|}{q - |Y_i|}$$

where

$$\omega = \max_{\lambda \in Y_i} r_i(\lambda)$$

Ranking Loss is the average percentage of incorrect labels that are ranked higher than correct labels.

$$RankingLoss = \frac{1}{m} \sum_{i=1}^m \frac{1}{|Y_i||\bar{Y}_i|} |\{(\lambda_a, \lambda_b) : r_i(\lambda_a) > r_i(\lambda_b), (\lambda_a, \lambda_b) \in Y_i \times \bar{Y}_i\}|$$

Each of these seven metrics tell us different things about the performance of the multi-label algorithm. In order to provide a single metric that at least partially captures all these different aspects, an additional metric is added that averages the previous seven together. For the metrics that are to be minimized instead of maximized, one minus the value is done in the averaging. This metric is called the *Overall Average*.

When comparing one learning model to another, it is important to have confidence that one model truly outperforms the other. One way to do this is to measure how statistically significant the difference is between their performance on each metric. This significant difference can be measured using the *paired permutation test*. The paired permutation test outputs a *p-value* between 0 and 1; the higher the *p-value*, the less statistically significant the difference is between two models. It is common convention that a *p-value* less than 0.05 means that it can be said with confidence that one model outperforms the other. The paired permutation test can only compare two models at a time; hence, for each metric, the paired permutation test is only calculated between the BAIN algorithm and the next best performing algorithm.

Chapter 5

Results

This chapter presents the results of the four experiments described in Chapter 4. The first experiment involves artificially removing an incremental percentage of labels in the training data. The second experiment uses the DARCI data set, which is a real world problem with a high percentage of missing labels. The third experiment considers the case where each training instance has exactly one positive label with all other labels unknown. The fourth experiment evaluates how well the BAIN algorithm can learn incrementally. Finally, the on-line multi-label Bayesian algorithm described in Section 2.3 is compared to the BAIN algorithm.

5.1 Artificially Removing Labels

The first experiment involves the ten data sets described in Section 4.1. Each algorithm is run on each data set using 10-fold cross validation. For each fold, a certain percentage of the labels are artificially removed from the training data. The model is trained with this modified training data and then evaluated using test data that still has all the labels present. The percentage of labels that are removed changes in 10% increments from 0% to 90%. An additional run at 95% is also performed because 95% is close to the percentage of missing labels that exists in the DARCI data set. The results from all ten data sets are averaged for each algorithm and for each evaluation metric. The results for overall average, accuracy, precision and recall can be seen in Figure 5.1. The results for Hamming loss, one-error, coverage, and ranking loss can be seen in Figure 5.2. The paired permutation test was

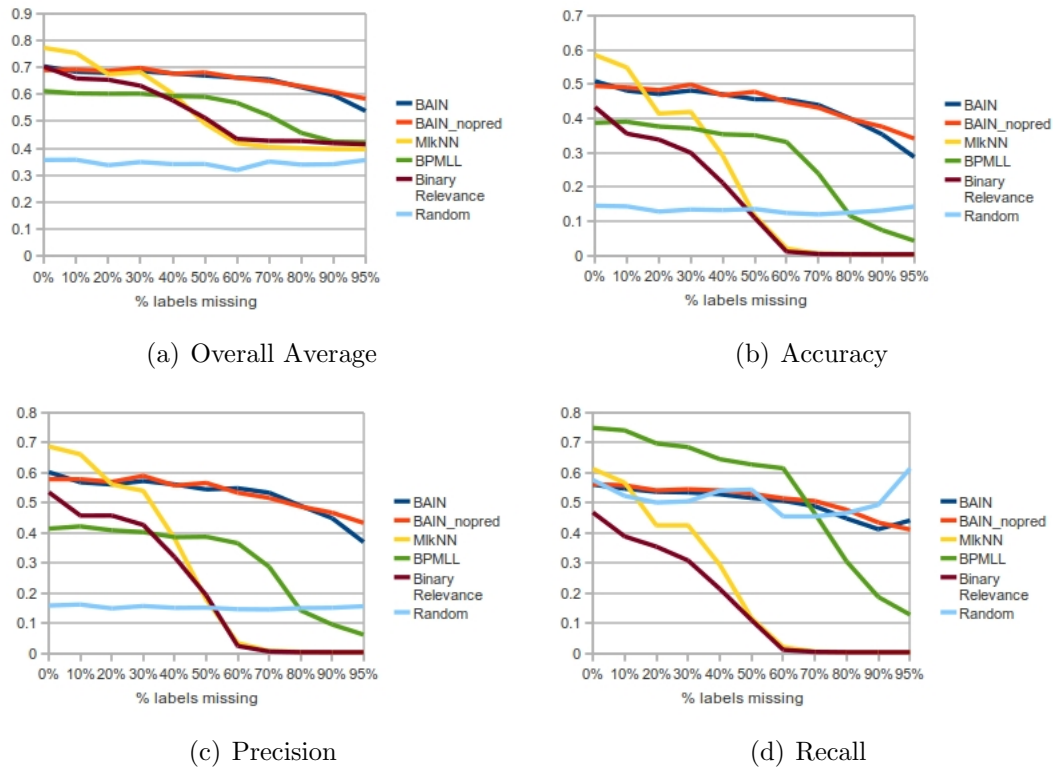


Figure 5.1: Average results of 10 different data sets as the percentage of missing labels increases (higher is better for these metrics). As more labels are removed, the performance of the other algorithms decreases rapidly, while the BAIN algorithm appears much more robust to missing labels and its performance decreases much more slowly.

performed for each of these metrics between the best version of BAIN and the next best algorithm (non-BAIN). The paired permutation test results for each metric can be seen in Figures 5.3 and 5.4.

When all labels are present, the BAIN algorithm does well, but is not the best. However, as labels are removed, the performance of the other algorithms decreases rapidly, while the BAIN algorithm appears much more robust to missing labels and its performance decreases much more slowly. In some cases, as with the MLkNN algorithm, when the percentage of labels missing reaches a threshold it stops predicting any labels as evidenced by the precision and recall dropping to zero. By the time 95% of the labels are missing, the BAIN algorithm outperforms all other algorithms in almost every metric, and does so by a considerable margin with accuracy, precision, and recall as shown in Figure 5.1.

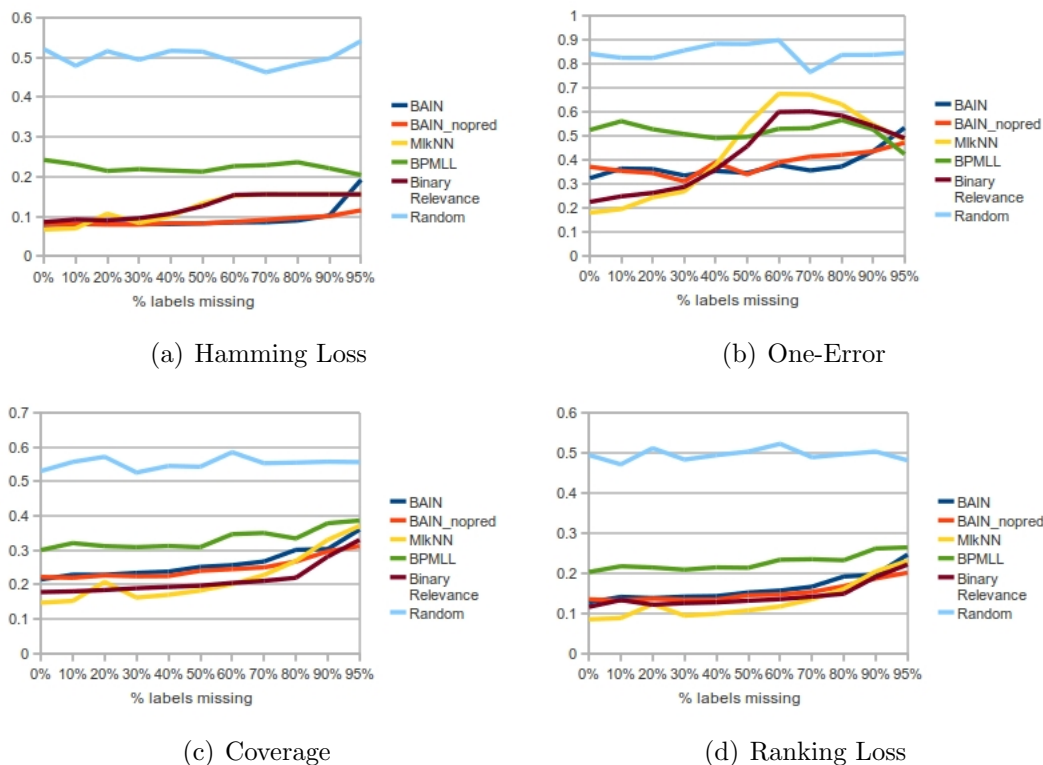


Figure 5.2: Average results of 10 different data sets as the percentage of missing labels increases (lower is better for these metrics). The BAIN algorithm outperforms almost all other algorithms by the time 95% of the labels are missing.

The paired permutation test results (Figures 5.3 and 5.4) show the best BAIN algorithm compared to the best non-BAIN algorithm. The red line indicates that the other algorithm is performing better than BAIN, while the blue line indicates that BAIN is performing better. When all labels are present, the p -value is below 0.05 and red for all metrics, which means that the BAIN algorithm starts off performing worse than the next best algorithm. However, as the percentage of missing labels increases, the p -values also increase, which means there is less and less confidence in the difference between BAIN and the next best algorithm. At some threshold the p -values change to blue, indicating that BAIN is now the best performing algorithm. This transition can be seen in Figures 5.1 and 5.2 when the BAIN algorithm crosses with the next best algorithm.

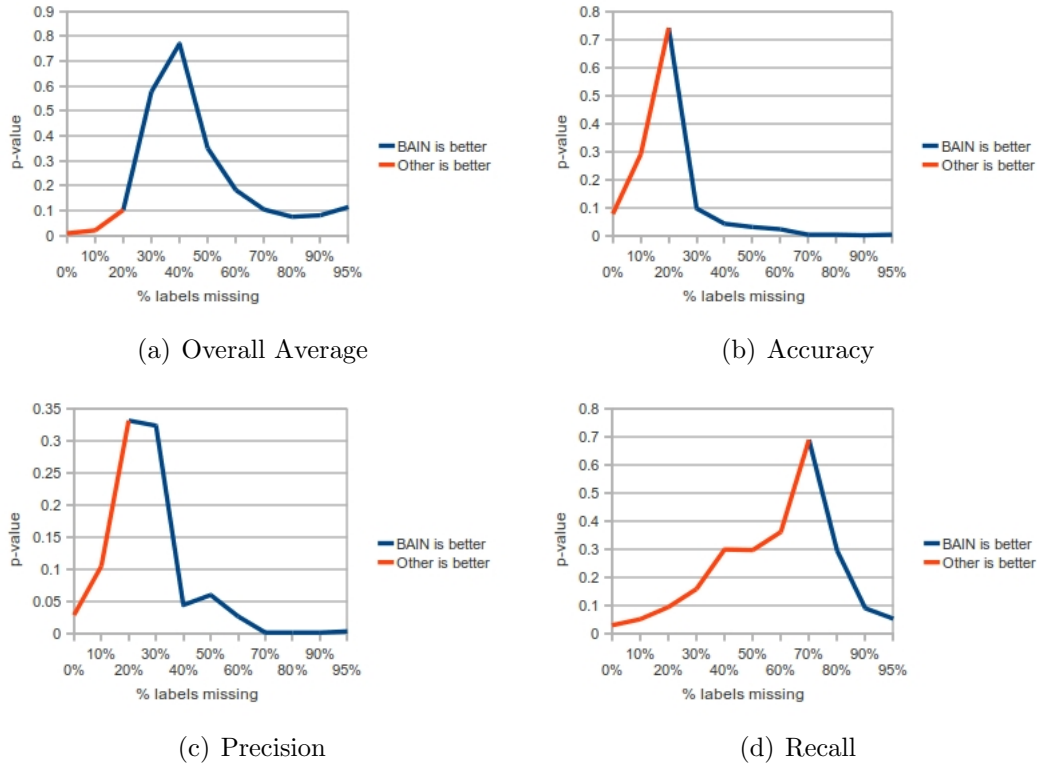


Figure 5.3: Paired permutation test results between BAIN and the next best algorithm corresponding to the metrics in Figure 5.1. When 95% of the labels are missing, the p -value is below 0.05 for accuracy, precision and recall, which means that there is high confidence that the difference between BAIN and the next best algorithm is statistically significant for those metrics.

By the time 95% of the labels are missing, the p -values are blue and less than 0.05 for accuracy, precision, and recall. This means that we can confidently say that the BAIN algorithm outperforms the other algorithms for those metrics. For the ranking-based metrics and Hamming loss, however, the p -values are still quite high. This means that, even though the BAIN algorithm is performing better on those metrics, there is still uncertainty in how significant that difference is. This uncertainty can be expected for ranking-based metrics because a complete ranking is much more challenging to get correct than the binary prediction of labels. With coverage and ranking loss, that uncertainty is more prominent, which is why their p -values take longer to change to blue. With Hamming loss and one-error, the p -values drop close to 0.05 at around 50% - 60% missing labels with BAIN performing better; however,

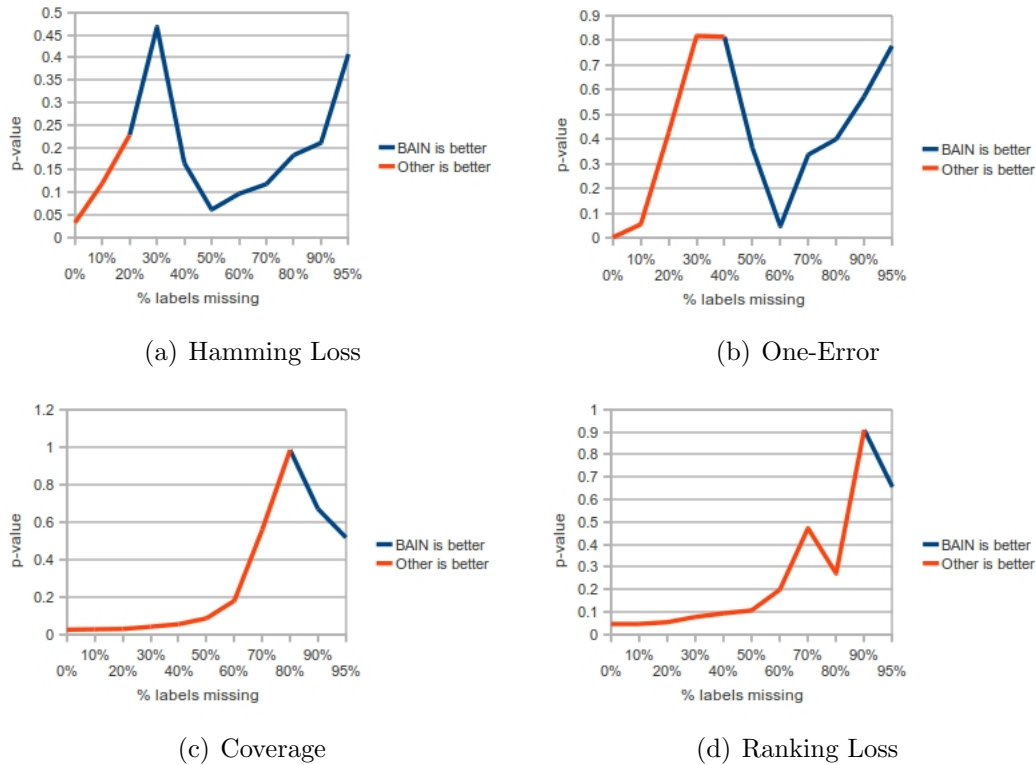


Figure 5.4: Paired permutation test results between BAIN and the next best algorithm corresponding to the metrics in Figure 5.2. The p -value is never below 0.05 when BAIN is better, which means that there is little confidence in how significant the difference is between BAIN and the next best algorithm for these metrics.

the p -values then go back up. This indicates that, for Hamming loss and one-error, there is a certain range of missing labels that BAIN confidently performs better at. However, when too many labels are missing, there is no longer adequate confidence that the BAIN algorithm truly performs better.

It is important to note that there is little difference in performance by the BAIN algorithm compared to the BAIN algorithm without the naïve Bayes label prediction (BAIN_nopred). These results show that our naïve Bayes method for inferring missing labels neither helps nor hurts our performance. In practice, it would therefore be better to not include the naïve Bayes approach as it requires more processing time, without any benefit.

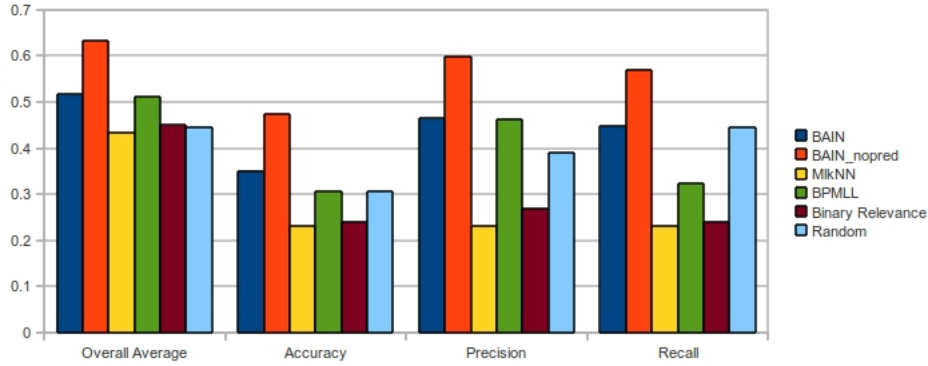


Figure 5.5: Results of the DARCI data set for overall average, accuracy, precision, and recall (higher is better for these metrics). BAIN without label prediction performs better on every metric than all the other algorithms.

5.2 The DARCI Data Set

The second experiment involves the DARCI data set, which is a real world problem that has missing labels. As mentioned before, approximately 95.3% of the labels are missing in the DARCI data set. This makes evaluation more challenging because the test sets used at each fold are also missing labels. The evaluation metrics can only be applied to labels that are actually known. While this limitation does not give us a complete measure of performance, it stills gives us general insight into how well each algorithm performs relative to one another. The results for overall average, accuracy, precision, and recall can be seen in Figure 5.5. The results for Hamming loss, one-error, coverage, and ranking loss can be seen in Figure 5.6. The paired permutation test was performed between the best BAIN and the best non-BAIN algorithm and the results for each metric can be seen in Table 5.1.

The results show that BAIN without label prediction is the clear winner in every metric. Normal BAIN and BPMLL are the next best and perform equally well overall with BPMLL doing better on Hamming loss and one-error and BAIN doing better on accuracy and recall. MLkNN and binary relevance, however, perform no better than random overall. All algorithms do better than random with Hamming loss and all the ranking based metrics. The paired permutation test results show p -values less than 0.05 for all metrics, which means

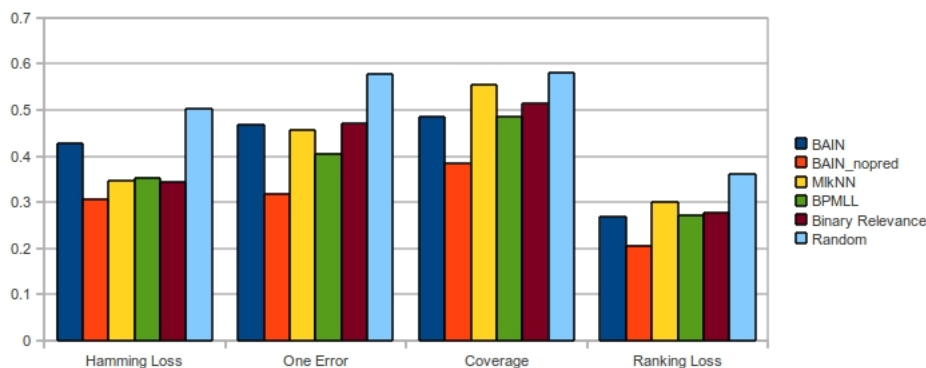


Figure 5.6: Results of the DARCI data set for Hamming loss, one-error, coverage, and recall (lower is better for these metrics). BAIN without label prediction performs better on every metric than all the other algorithms.

Table 5.1: Paired permutation test results between the best BAIN and the best non-BAIN algorithm for the DARCI data set corresponding to Figures 5.5 and 5.6. The p -values are below 0.05 for all metrics, which means that there is high confidence that the difference between BAIN and the next best algorithm is statistically significant.

	Overall Average	Accuracy	Precision	Recall
p -value	0.00098	0.00098	0.00098	0.00098

	Hamming Loss	One-Error	Coverage	Ranking Loss
p -value	0.00293	0.00098	0.00098	0.00098

that the difference in performance between BAIN without label prediction and the next best algorithm is statistically significant.

With the DARCI data set, it appears that inferring missing labels hurts the performance of the BAIN algorithm, whereas in the previous experiment it didn't seem to affect the performance. A reason for this could be due to the previously mentioned limitation of not having a test set with no missing labels. A full test set may reveal that label prediction makes no difference for the DARCI data set. A more likely explanation, however, could be the fact that our naïve Bayes method for inferring missing labels relies on at least some of the labels being present in order to make any kind of accurate inference on the labels that are missing. Having only 5% of the labels present in the data simply may not be enough for an accurate prediction and hence many of the predictions are inaccurate and the resulting

performance is worse. There is evidence from the previous experiment to support this. If we refer back to Figure 5.1 and Figure 5.2, we can see that the performance of BAIN at 95% missing labels is worse than BAIN without label prediction in every metric except recall.

5.3 Only One Positive Label

The third experiment involves the specific case where each training instance has exactly one positive label with all other labels being unknown. This allows us to compare the BAIN algorithm to the WIN algorithm, which was designed for this specific case, in addition to the other algorithms previously used. Using the same ten data sets from Section 4.1 and 10-fold cross validation, we artificially remove all labels except for one randomly chosen positive label from each instance of the training data. The algorithms are then evaluated using the fully labeled test set. This experiment will only use the version of BAIN without label prediction as it was shown in the previous two experiments to perform better than BAIN with label prediction. For further comparison, the BAIN algorithm will also be run on the ten data sets where a randomly chosen negative label is provided for each instance in addition to the positive label. The results from all ten data sets are averaged together for each algorithm and for each metric. The results for overall average, accuracy, precision, and recall can be seen in Figure 5.7, The results for Hamming loss, one-error, coverage, and ranking loss can be seen in Figure 5.8. The paired permutation test results between BAIN with one positive label and the best non-BAIN algorithm can be seen in Table 5.2. The paired permutation test results between the best non-BAIN algorithm and BAIN with one positive and one negative label can be seen in Table 5.3.

The results show that the BAIN algorithm, although better than random, performs worse than the other algorithms for all metrics except for recall. This experiment exposes BAIN's main weakness of relying on at least some explicitly given negative labels being present in the training data. The other algorithms do not suffer from this same weakness as they automatically assume any missing label to be negative. However, if at least one

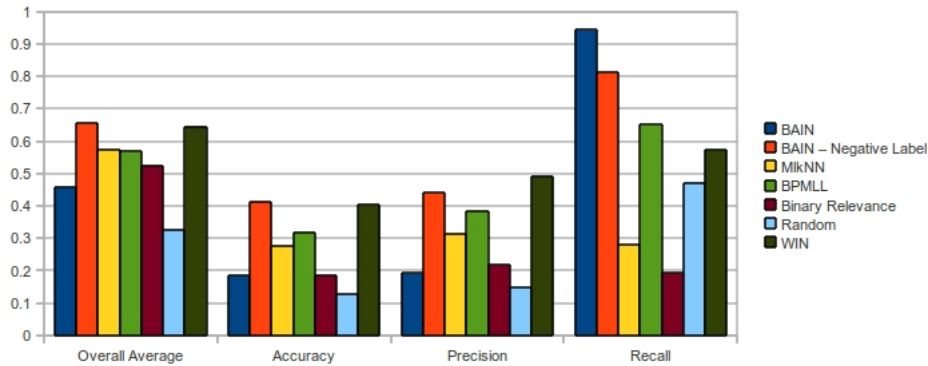


Figure 5.7: Results for overall average, accuracy, precision, and recall when there is only one positive label for each training instance (higher is better for these metrics). With the exception of recall, BAIN performs worse than the other algorithms. However, when BAIN is given at least one negative label per training instance, it performs comparable to the WIN algorithm.

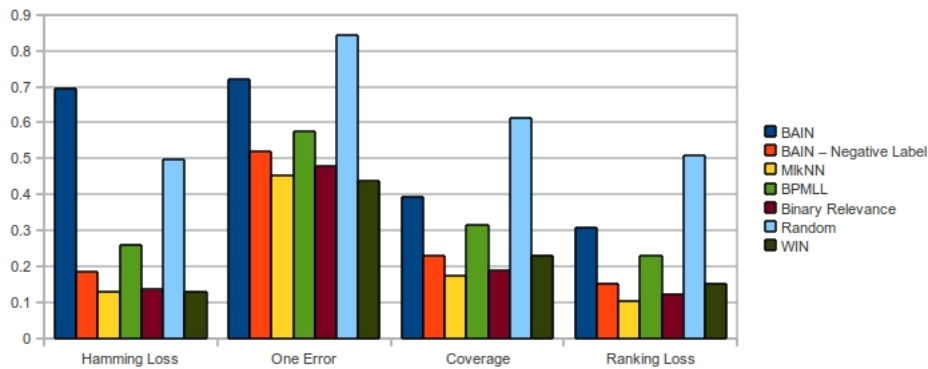


Figure 5.8: Results of Hamming loss, one-error, coverage and ranking loss when there is only one positive label for each training instance (lower is better for these metrics). BAIN performs worse than the other algorithms. However, when at least one additional negative label per training instance is given to BAIN, its performance increases to be competitive with the other algorithms.

Table 5.2: Paired permutation test results between BAIN and the best non-BAIN algorithm corresponding to Figures 5.7 and 5.8. All the p -values are less than 0.05, which means that there is high confidence that the difference between BAIN and the best non-BAIN algorithm is statistically significant.

	Overall Average	Accuracy	Precision	Recall
p-value	0.00098	0.00684	0.00293	0.00293

	Hamming Loss	One-Error	Coverage	Ranking Loss
p-value	0.00098	0.00684	0.00293	0.00293

Table 5.3: Paired permutation test results between BAIN with a negative label and the best non-BAIN algorithm corresponding to Figures 5.7 and 5.8. With the exception of recall, none of the p -values are less than 0.05, which means that there is little confidence that the difference between BAIN with a negative label and the best non-BAIN algorithm is statistically significant.

	Overall Average	Accuracy	Precision	Recall
p-value	0.49902	0.70996	0.21973	0.01660

	Hamming Loss	One-Error	Coverage	Ranking Loss
p-value	0.15723	0.10645	0.13184	0.06934

negative label per training instance is present, then the performance of BAIN improves considerably and is comparable to the WIN algorithm. The paired permutation test results in Table 5.3 show no significant difference between BAIN with a single negative label and the best non-BAIN algorithm for all metrics except recall, where BAIN is superior. Adding an explicit negative label makes no difference to the other algorithms because that label is already assumed to be negative. In general, explicit negative labels are harder to acquire than positive labels. However, it may not be unreasonable to require that there be at least one negative label provided for each training instance when using the BAIN algorithm. This is because there are usually far more negative labels per instance than positive labels. Table 4.1 shows that the average density of all the data sets we have used is 0.143, which means that only 14.3% of the possible labels are positive, while the remaining 85.7% are negative.

5.4 Incremental Learning

The fourth experiment is designed to evaluate the incremental learning capabilities of the BAIN algorithm and again involves 10-fold cross validation and the ten data sets described in Section 4.1. The incrementally trained model is compared with a model that was trained with the entire data set from the start. In order for the performance of the two models to be comparable, the incrementally trained model is trained so that it eventually sees all the same data that the other model had from the beginning.

To help illustrate, consider an example with the possible labels $L = \{A, B, C, D\}$ and training set:

$$T = \begin{cases} x_1 \rightarrow A, \bar{B}, \bar{C}, \bar{D} \\ x_2 \rightarrow \bar{A}, \bar{B}, C, D \\ x_3 \rightarrow A, B, \bar{C}, \bar{D} \\ x_4 \rightarrow \bar{A}, \bar{B}, C, \bar{D} \end{cases}$$

First, the training data is shuffled and split in half.

$$T_1 = \begin{cases} x_2 \rightarrow \bar{A}, \bar{B}, C, D \\ x_3 \rightarrow A, B, \bar{C}, \bar{D} \end{cases}$$

$$T_2 = \begin{cases} x_1 \rightarrow A, \bar{B}, \bar{C}, \bar{D} \\ x_4 \rightarrow \bar{A}, \bar{B}, C, \bar{D} \end{cases}$$

Some labels are randomly chosen and removed from each instance of the first half of the training data (the same labels are chosen for each fold). (In the actual experiment, we removed 5 labels).

$$T_1 = \begin{cases} x_2 \rightarrow \bar{A}, \bar{B} \\ x_3 \rightarrow A, B \end{cases}$$

$$T_2 = \begin{cases} x_1 \rightarrow A, \bar{B}, \bar{C}, \bar{D} \\ x_4 \rightarrow \bar{A}, \bar{B}, C, \bar{D} \end{cases}$$

Each instance of the first half of the training data is then added to the second half with only the removed labels provided. This is to ensure that the model will eventually see the labels that were removed.

Table 5.4: Paired permutation test results between the BAIN that has been trained incrementally and the BAIN that has been trained with all the data from the start corresponding to Figures 5.9 and 5.10. All the p -values are high, indicating that there is no significant difference between their performances.

	Overall Average	Accuracy	Precision	Recall
p-value	0.55176	0.72363	0.50293	0.31348

	Hamming Loss	One-Error	Coverage	Ranking Loss
p-value	0.62402	0.26465	0.55567	0.09277

$$T_1 = \begin{cases} x_2 \rightarrow \bar{A}, \bar{B} \\ x_3 \rightarrow A, B \end{cases}$$

$$T_2 = \begin{cases} x_1 \rightarrow A, \bar{B}, \bar{C}, \bar{D} \\ x_4 \rightarrow \bar{A}, \bar{B}, C, \bar{D} \\ x_2 \rightarrow C, D \\ x_3 \rightarrow \bar{C}, \bar{D} \end{cases}$$

The BAIN algorithm is trained using T_1 and is unaware that the removed labels exist. After initial training, the model is then trained with T_2 . The removed labels are introduced to the model at the same time and the model must learn incrementally as described in Section 3.3.

This incrementally trained model is then compared with a model that was trained with the entire data set from the start. A version of BAIN that does not learn incrementally is also used as a baseline comparison. Most multi-label learning algorithms that do not handle new incoming labels will either crash, or ignore the new label. This version of BAIN just ignores any new labels during training. During evaluation, the removed labels are never predicted and are always ranked at the bottom. The results from all ten data sets are averaged for each model. The results for overall average, accuracy, precision and recall can be seen in Figure 5.9. The results for Hamming loss, one-error, coverage, and ranking loss can be seen in Figure 5.10. The paired permutation test results between the incrementally trained BAIN and the BAIN trained with all the data from the start can be seen in Table 5.4. The paired permutation test results between the incrementally trained BAIN and the non-incremental baseline BAIN can be seen in Table 5.5.

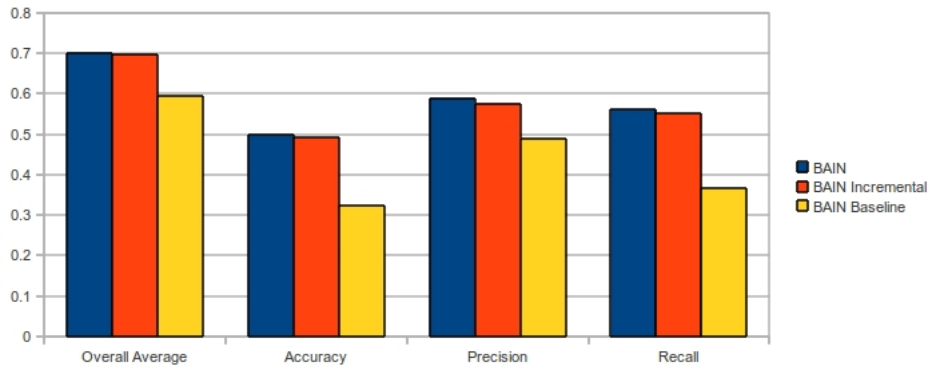


Figure 5.9: Results of incremental training versus non-incremental for overall average, accuracy, precision, and recall (higher is better for these metrics). The BAIN that has been trained incrementally performs comparable to the BAIN that has been trained with all the data from the start.

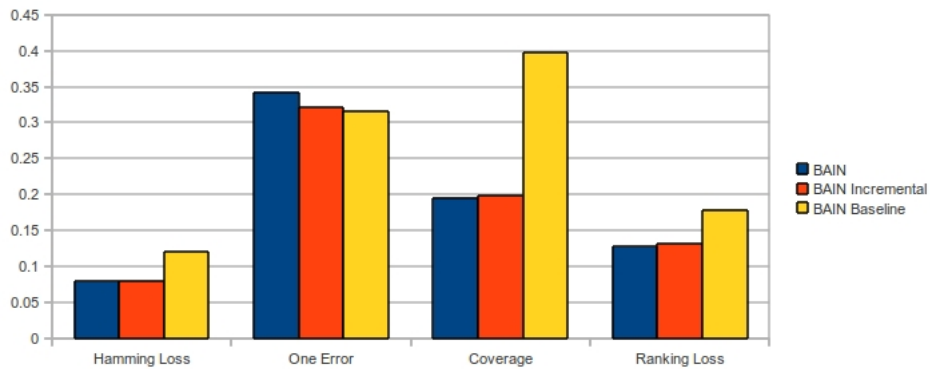


Figure 5.10: Results of incremental training versus non-incremental for Hamming loss, one-error, coverage, and recall (lower is better for these metrics). The BAIN that has been trained incrementally performs comparable to the BAIN that has been trained with all the data from the start.

Table 5.5: Paired permutation test results, corresponding to Figures 5.9 and 5.10, between the BAIN that has been trained incrementally and the baseline BAIN that cannot handle new incoming labels. With the exception of one-error and ranking loss, the p -values are low, which means there is high confidence that the difference between these two models is statistically significant.

	Overall Average	Accuracy	Precision	Recall
p-value	0.03613	0.03223	0.10254	0.08691

	Hamming Loss	One-Error	Coverage	Ranking Loss
p-value	0.07324	0.89941	0.00488	0.24707

As expected, the results show little significant difference in performance between the model that was trained incrementally and the model that was trained with all the data from the start. The BAIN algorithm can handle new incoming labels during incremental training, and the resulting model achieves similar performance to a model trained with the new labels from the beginning. The non-incremental baseline model performs worse than the incremental model for every metric except one-error. The paired permutation test results for these two models in Table 5.5 show a significant difference for all metrics except for one-error, ranking loss, recall, precision, and Hamming loss, although Hamming loss, precision, and recall are close to 0.05. The reason one-error actually performs slightly better is because one-error only measures when the top ranked label is a false positive. The five removed labels are never given the chance to be false positives and are never ranked at the top and hence, never count against one-error. With the exception of one-error, the results clearly show that accounting for new incoming labels during incremental training significantly improves the performance.

5.5 Comparing with the Bayesian Model

The on-line Bayesian multi-label learning algorithm [21] discussed in Section 2.3 was not included in the previous experiments due to its strictly on-line learning method and its inefficiency on problems with a large number of possible labels. However, in their paper they use the yeast and scene data sets, which are two of the data sets used in our experiments (see

Table 4.1). We can, therefore, compare our results on those data sets with the results they achieved in their paper. The authors use the $F1$ score to evaluate their algorithm, which is defined as:

$$F1 = \frac{2pr}{p + r}$$

where p and r are precision and recall respectively.

Using their Bayesian active learning approach on the yeast data set, they were able to achieve an $F1$ score of 0.58. The BAIN algorithm, using ten-fold cross validation with all the labels provided, achieved an $F1$ score of 0.64. However, on the scene data set, they had an $F1$ score of 0.91, while the BAIN algorithm only achieved 0.70. Clearly there are trade-offs between the two algorithms. The results indicate that the Bayesian algorithm may be better for data sets with a small number of possible labels, while the BAIN algorithm can better handle problems with a large number of possible labels. The Bayesian algorithm is strictly an on-line learner, so if the training has to be restarted, then it can become intractable to retrain the model with the whole training set. The BAIN algorithm is more flexible as it can learn incrementally or with all the data at once.

Chapter 6

Conclusions and Future Work

The BAIN algorithm is successful at improving multi-label classification performance for problems where the labeling is incomplete. We have shown that, compared to other common multi-label algorithms, the BAIN algorithm is more robust in performance as the percentage of missing labels increases. On the DARCI data set, which is a real world problem that has missing labels, we have shown that the BAIN algorithm successfully performs better than other multi-label learning algorithms. We have also shown that the BAIN algorithm can learn incrementally and handle new labels that were previously unknown with hardly any loss of performance.

The naïve Bayes method to infer missing labels was shown to be ineffective and even detrimental in the case of the DARCI data set. Either the labels that are inferred are redundant and make little difference, or there are not enough provided labels to accurately infer the missing labels. In either case, it is clear that the BAIN algorithm without this label inference method is the better choice. Further research would be useful in discovering other methods that could be more effective than the naïve Bayes approach.

The BAIN algorithm was shown to perform poorly when there are no explicitly given negative labels in the training data. However, one negative label provided per training instance is enough to significantly increase BAIN's performance to be competitive with the WIN algorithm. This requirement may not be unreasonable as there are far more possible negative labels than positive labels for each training instance. Additional research needs to be done on multiple real-world data sets with missing labels to better determine how plausible

that requirement is. The WIN algorithm performed well when only one positive label was provided per training instance. Additional research could extend the WIN algorithm to handle data sets with any number of provided labels, both positive and negative. The BAIN algorithm could be extended to overcome its weakness of requiring at least some explicitly given negative labels. However, there would likely be a trade-off because the point of the BAIN algorithm is to avoid assumptions about missing data.

The BAIN algorithm was shown to have trade-offs compared to the on-line multi-label Bayesian model. The BAIN algorithm can handle problems with a large number of possible labels, while the Bayesian model is inefficient on these problems. This is likely because the Bayesian algorithm explicitly attempts to model the higher order relationships between labels. While this is a significant advantage on problems with a small number of possible labels (such as the scene data set), it is too slow for large sets of possible labels. Additional research needs to be done to see how the BAIN algorithm could benefit from trying to model these higher order relationships.

Removing the assumption of implicit negativity is effective in improving multi-label classification performance for the backpropagation algorithm. Other multi-label algorithms, such as MLkNN, could be modified to remove that assumption, or adapted in other ways to handle missing labels. Instead of trying to infer the missing labels for each training instance, active learning approaches could be used to provide additional training instances for the labels not adequately represented in the current data set. There are many opportunities for future research in this area; the BAIN algorithm provides a stepping stone as a simple yet effective solution to problems with missing labels.

References

- [1] Zhang, M.L., Zhou, Z.H.: Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering* **18**(10) (2006) 1338–1351
- [2] Schapire, R.E., Singer, Y.: BoosTexter: A boosting-based system for text categorization. *Machine Learning* **39**(2/3) (2000) 135–168
- [3] Boutell, M.R., Luo, J., Shen, X., Brown, C.M.: Learning multi-label scene classification. *Pattern Recognition* **37**(9) (2004) 1757–1771
- [4] Tsoumakas, G., Katakis, I., Vlahavas, I.: Effective and efficient multilabel classification in domains with large number of labels. In: *Proceedings ECML/PKDD Workshop on Mining Multidimensional Data*. (2008)
- [5] Trohidis, K., Tsoumakas, G., Kalliris, G., Vlahavas, I.: Multilabel classification of music into emotions. In: *Proceedings of the 9th International Conference on Music Information Retrieval*. (2008) 325–330
- [6] Tsoumakas, G., Katakis, I.: Multi-label classification: An overview. *International Journal of Data Warehousing and Mining* **3**(3) (2007) 1–13
- [7] Zhang, M.L., Zhou, Z.H.: A k-nearest neighbor based algorithm for multi-label classification. In: *Proceedings of the IEEE International Conference on Granular Computing*. (2005) 718–721
- [8] Tsoumakas, G., Katakis, I., Vlahavas, I.P.: Mining multi-label data. In: *Data Mining and Knowledge Discovery Handbook*. (2010) 667–685
- [9] de Carvalho, Freitas, A.A. In: *A tutorial on multi-label classification techniques*. Volume 205 of *Studies in Computational Intelligence*. Springer (2009) 177–195
- [10] Clare, A., King, R.D.: Knowledge discovery in multi-label phenotype data. In: *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, Springer-Verlag (2001) 42–53

- [11] Godbole, S., Sarawagi, S.: Discriminative methods for multi-labeled classification. *Advances in Knowledge Discovery and Data Mining* (2004) 22–30
- [12] Zhang, M.L.: ML-RBF: RBF neural networks for multi-label learning. *Neural Processing Letters* **29**(2) (2009) 61–74
- [13] Taskar, B., Guestrin, C., Koller, D.: Max-margin markov networks. In: *Advances in Neural Information Processing Systems*. (2003) 8–13
- [14] Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y.: Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research* **6** (2005) 1453–1484
- [15] Bielza, C., Li, G., Larraòaga, P.: Multi-dimensional classification with bayesian networks. *International Journal of Approximate Reasoning* **52** (2011) 705–727
- [16] Bakir, G.H., Hofmann, T., Schölkopf, B., Smola, A.J., Taskar, B., Vishwanathan, S.V.N.: *Predicting Structured Data*. The MIT Press (2007)
- [17] Whiting, S., Ventura, D.: Learning multiple correct classifications from incomplete data using weakened implicit negatives. In: *Proceedings of the International Joint Conference on Neural Networks*. (2004) 2953–2958
- [18] Tsuboi, Y., Kashima, H., Mori, S., Oda, H., Matsumoto, Y.: Training conditional random fields using incomplete annotations. In: *International Conference on Computational Linguistics*. (2008) 897–904
- [19] Vishwanathan, A.S., Smola, A.J., Vishwanathan, S.V.N.: Kernel methods for missing variables. In: *In Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*. (2005) 325–332
- [20] Suzuki, J., Fujino, A., Isozaki, H.: Semi-supervised structured output learning based on a hybrid generative and discriminative approach. In: *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. (2007) 791–800
- [21] Qi, G.J., Xian-Sheng, Rui, Y., Tang, J., Zhang, H.J.: Two-dimensional multi-label active learning with an efficient online adaptation model for image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **31** (2009) 1880–1897

- [22] Zhu, S., Ji, X., Xu, W., Gong, Y.: Multi-labelled classification using maximum entropy method. In: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM Press (2005) 274–281
- [23] Guo, X., Yin, Y., Dong, C., Yang, G., Zhou, G.: On the class imbalance problem. In: Proceedings of the Fourth International Conference on Natural Computation, IEEE Computer Society (2008) 192–201
- [24] Guo, H., Viktor, H.L.: Learning from imbalanced data sets with boosting and data generation: the DataBoost-IM approach. SIGKDD Explorations Newsletter **6**(1) (2004) 30–39
- [25] Han, H., Wang, W.Y., Mao, B.H.: Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning. In Huang, D.S., Zhang, X.P., Huang, G.B., eds.: Advances in Intelligent Computing. Volume 3644 of Lecture Notes in Computer Science., Springer (2005) 878–887
- [26] Chen, K., Lu, B.: Efficient classification of multilabel and imbalanced data using min-max modular classifiers. In: Proceeding of the International Joint Conference on Neural Networks. (2006) 1770–1775
- [27] Bruzzone, L., Fernández Prieto, D.: An incremental-learning neural network for the classification of remote-sensing images. Pattern Recognition Letters **20** (1999) 1241–1248
- [28] Polikar, R., Udpa, L., Udpa, S., Member, S., Member, S., Honavar, V.: Learn++: An incremental learning algorithm for supervised neural networks. IEEE Transactions on Systems, Man and Cybernetics (C), Special Issue on Knowledge Management **31**(4) (2001) 497–508
- [29] Hua, X.S., Qi, G.J.: Online multi-label active annotation: Towards large-scale content-based video search. In: Proceedings of the 16th ACM International Conference on Multimedia, ACM (2008) 141–150
- [30] Tsoumakas, G., Spyromitros-Xioufis, E., Vilcek, J., Vlahavas, I.: Mulan: A java library for multi-label learning. Journal of Machine Learning Research **12** (2011) 2411–2414
- [31] Norton, D., Heath, D., Ventura, D.: Establishing appreciation in a creative system. In: Proceedings of the International Conference on Computational Creativity. (2010) 26–35