Deakin Research Online

This is the published version:

Kouzani, Abbas Z. and Nasireding, Gulisong 2009, Multilabel classification by BCH code and random forests, *International journal of recent trends in engineering*, vol. 2, no. 1, pp. 113-116.

Available from Deakin Research Online:

http://hdl.handle.net/10536/DRO/DU:30028670

Reproduced with the kind permission of the copyright owner.

Copyright: 2009, Academy Publisher.

International Journal of Recent Trends in Engineering, Vol 2, No. 1, November 2009

Multilabel Classification by BCH Code and Random Forests

Abbas Z. Kouzani and Gulisong Nasireding
School of Engineering, Deakin University, Geelong,VIC 3217, Australia
Email: {kouzani, gn}@deakin.edu.au

***Abstract*—Multilabel classification deals with problems in which an instance can belong to multiple classes. This paper uses error correcting codes for multilabel classification. BCH code and Random Forests learner are used to form the proposed method. Thus, the advantage of the error-correcting properties of BCH is merged with the good performance of the random forests learner to enhance the multilabel classification results. Three experiments are conducted on three common benchmark datasets. The results are compared against those of several exiting approaches. The proposed method does well against its counterparts for the three datasets of varying characteristics.**

***Index Terms*—multilabel data, multilabel classification, error correcting codes, BCH code, ensemble learners, random forests**

I. Introduction

Singlelabel classification refers to learning from a collection of instances that each is related to only one label l from a set of labels L. Multilabel classification, on the other hand, refers to learning from a set of instances that each is associated with a set of labels $Y \subseteq L$ [1]. A sample multilabel dataset is shown is Table I. It consists of three instances. Each instance contains four features. There are three classes $L = \{C1,C2,C3\}$. Each instance belongs to one class or multiple classes.

TABLE I.
SAMPLE MULTILABEL DATASET

Inst	Features				C1	C2	C3
1	F_{11}	F_{12}	F_{13}	F_{14}	0	1	1
2	F_{21}	F_{22}	F_{13}	F_{24}	1	1	0
3	F_{31}	F_{32}	F_{13}	F_{34}	0	0	1

Multilabel classification has received some attentions in the past several years. A number of methods have been developed to tackle multilabel classification problems. Some key methods are reviewed in the following.

According to Brinker et al. [2], Binary Relevance (BR) considers the prediction of each label as an independent binary classification task. It trains a separate binary relevance model for each possible label using all examples related to the label as positive examples and all other examples as negative examples. For classifying a new instance, all binary predictions are obtained and then the set of labels corresponding to positive relevance classification is associated with the instance.

Zhang and Zhou [3] report a multi-label lazy learning approach which is derived from the traditional k-Nearest Neighbour (kNN) and named ML-kNN. For each unseen instance, its k nearest neighbors in the training set are identified. Then, based on statistical information gained from the label sets of these neighboring instances, maximum a posteriori principle is used to determine the label set for the unseen instance.

Zhang and Zhou [4] present a neural network-based algorithm that is Backpropagation for Multi-Label Learning named BP-MLL. It is based on the backpropagation algorithm but uses a specific error function that captures the characteristics of multi-label learning. The labels belonging to an instance are ranked higher than those not belonging to that instance.

Tsoumakas and Vlahavas [5] propose RAndom K-labELsets (RAKEL) which is an ensemble method for multilabel classification based on random projections of the label space. An ensemble of Label Powerset (LP) classifiers is trained on smaller size of label subset randomly selected from the training data. The RAKEL takes into account label correlations by using single-label classifiers that are applied on subtasks with manageable number of labels and adequate number of examples per label. It therefore tackles difficulty of learning due to a large number of classes associated with only a few examples.

Several other important works can be also found in [6-11]. The main motivation behind the work reported in this paper is our desire to improve the performance of the multilabel classification methods. This paper explores the use of error correcting code for multilabel classification. It uses the Bose, Ray-Chaudhuri, Hocquenghem (BCH) code and Random Forests learner to form a method that can deal with multilabel classification problems improving the performance of several popular exiting methods. The description of the theoretical framework as well as the proposed method is given in the following sections.

II. Bose, Ray-Chaudhuri, Hocquenghem Code

Bose, Ray-Chaudhuri, Hocquenghem (BCH) Code is a multilevel, cyclic, error-correcting, variable-length digital code that can correct errors up to about 25% of the total number of digits [12-13]. The original applications of BCH code were limited to binary codes of length 2^m-1 for some integer m. These were extended later to the nonbinary codes with symbols from Galois field GF(q). Galois field is a field with a finite field order (number of elements). The order of a Galois field is always a prime or a power of a prime number. GF(q) is called the prime field of order q where the q elements are 0,1, …, q-1.

BCH codes are cyclic codes and can be specified by a generator polynomial. For any integer $m \geq 3$ and $t < 2^{m-1}$, there exists a primitive BCH code with the following parameters:

$$\begin{aligned} n &= 2^m - 1 \\ n - k &\leq mt \\ d_{min} &\geq 2t + 1 \end{aligned} \quad (1)$$

© 2009 ACADEMY PUBLISHER

The code can correct *t* or fewer random errors over a span of 2^m-1 bit positions. The code is called a t-error-correcting BCH code over GF(q) of length *n*. This code is specified as follows:

1. Determine the smallest *m* such that GF(q^m) has a primitive *n*th root of unity β.
2. Select a nonnegative integer *b*. Frequently, b=1.
3. Form a list of 2*t* consecutive powers of β: $\beta^b, \beta^{b+1}, \ldots, \beta^{b+2t-1}$. Determine the minimal polynomial with respect to GF(q) of each of these powers of β.
4. The generator polynomial *g(x)* is the least common multiple (LCM) of these minimal polynomials. The code is a *(n, n- deg(g(x)))* cyclic code.

Due to the fact that the generator is constructed using minimal polynomials with respect to GF(q), the generator *g(x)* has coefficients in GF(q), and the code is over GF(q). Two fields are involved in the construction of BCH codes. GF(q) is where the generator polynomial has its coefficients and is the field where the elements of the codewords are. GF(q^m) is the field where the generator polynomial has its roots. For encoding purpose, it is adequate to work only with GF(q). However, decoding requires operations in GF(q^m).

For binary BCH codes, let α be a primitive element in GF(2^m). For $1 \leq i \leq t$, let $\Phi_{2i-1}(x)$ be the minimum polynomial of the field element α^{2i-1}. The degree of $\Phi_{2i-1}(x)$ is *m* or a factor of *m*. The generator polynomial g(x) of t-error-correcting BCH codes of length 2^m-1 is given by:

$$g(x) = LCM\{\Phi_1(x), \Phi_3(x), \ldots, \Phi_{2t-1}(x)\} \quad (2)$$

The first explicit decoding algorithm for binary BCH codes was Peterson's algorithm that was useful only for correcting small numbers of errors. Berlekamp introduced the first truly efficient decoding algorithm for both binary and nonbinary BCH codes. This was further developed by Massey and is usually called the Berlekamp-Massey decoding algorithm.

Consider a BCH code with $n = 2^{m-1}$ and generator polynomial $g(x)$. Suppose a code polynomial $c(x) = c_0 + c_1 x + \ldots + c_{n-1} x^{n-1}$ is transmitted. Let $r(x) = r_0 + r_1 x + \ldots + r_{n-1} x^{n-1}$ be the received polynomial. Then, $r(x) = c(x) + e(x)$, where $e(x)$ is the error polynomial. To check whether $r(x)$ is a code polynomial, $r(\alpha) = r(\alpha^2) = \ldots = r(\alpha^{2t}) = 0$ is tested. If yes, then $r(x)$ is a code polynomial, otherwise $r(x)$ is not a code polynomial and the presence of errors is detected. The decoding procedure includes three steps: syndrome calculation, error pattern specification, and error correction.

III. Random Forests

Ensemble learning refers to the algorithms that produce collections of classifiers which learn to classify by training individual learners and fusing their predictions. Growing an ensemble of trees and getting them vote for the most popular class has given a good enhancement in the accuracy of classification. Random vectors are built that control the growth of each tree in the ensemble. The ensemble learning methods can be divided into two main groups: bagging and boosting.

In bagging, models are fit in parallel where successive trees do not depend on previous trees. Each tree is independently built using bootstrap sample of the dataset. A majority vote determines prediction. In boosting, models are fit sequentially where successive trees assign additional weight to those observations poorly predicted by previous model. A weighted vote specifies prediction.

A random forest [14] adds an additional degree of randomness to bagging. Although each tree is constructed using a different bootstrap sample of the dataset, the method by which the classification trees ate built is improved.

A random forest predictor is an ensemble of individual classification tree predictors. For each observation, each individual tree votes for one class and the forest predicts the class that has the plurality of votes. The user has to specify the number of randomly selected variables m_{try} to be searched through for the best split at each node. Whilst a node is split using the best split among all variables in standard trees, in a random forest the node is split using the best among a subset of predictors randomly chosen at that node. The largest tree possible is grown and is not pruned. The root node of each tree in the forest contains a bootstrap sample from the original data as the training set. The observations that are not in the training set are referred to as "out-of-bag" observations.

Since an individual tree is unpruned, the terminal nodes can contain only a small number of observations. The training data are run down each tree. If observations *i* and *j* both end up in the same terminal node, the similarity between *i* and *j* is increased by one. At the end of the forest construction, the similarities are symmetrised and divided by the number of trees. The similarity between an observation and itself is set to one. The similarities between objects form a matrix which is symmetric, and each entry lies in the unit interval *[0, 1]*. Breiman defines the random forest as [14]:

> A random forest is a classifier consisting of a collection of tree-structured classifiers $\{h(\mathbf{x}, \Theta_k), k = 1, \ldots\}$ where $\{\Theta_k\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input $\boldsymbol{x}$.

Fig. 1 displays a pseudo-code for the random forest algorithm. A summary of the random forest algorithm for classification is given below [15]:

- Draw *K* bootstrap samples from the training data.
- For each of the bootstrap samples, grow an unpruned classification tree, with the following modification: at each node, rather than choosing the best split among all predictors, randomly sample *m* of the predictors and choose the best split from among those variables.
- Predict new data by aggregating the predictions of the *K* trees, i.e., majority votes for classification, average for regression.

© 2009 ACADEMY PUBLISHER

- select the number of trees to be generated *K*
- for (*k* = 1; *k* <= *K*; *k*++)
  - draw a bootstrap sample Θ_k from the training data
  - grow an unpruned classification tree $h(\mathbf{x}, \Theta_k)$
  - for (*i* = 1; *i* <= *number-of-nodes*; *i*++)
    - randomly sample *m* predictor variables
    - select the best split from among those variables

  end

end

- each of the *K* classification trees casts 1 vote for the most popular class at input $\mathbf{x}$
- aggregate the classification of the *K* trees and select the class with maximum votes

Figure 1. Pseudo-code for the random forest algorithm.

The random forest approach works well because of: (i) the variance reduction achieved through averaging over learners, and (ii) randomised stages decreasing correlation between distinctive learners in the ensemble. The generalisation error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correlation between them. Using a random selection of features to split each node yields error rates that compare to AdaBoost [16] An estimate of the error rate can be obtained, based on the training data, by the following [15]:

- At each bootstrap iteration, predict the data that is not in the bootstrap sample, called ``out-of-bag" data, using the tree which is grown with the bootstrap sample.
- Aggregate the out-of-bag predictions. On the average, each data point would be out-of-bag around 36.8% [17] of the times. Calculate the error rate, and call it the "out-of-bag" estimate of error rate.

With regard to the 36.8%, the random forest forms a set of tree-based learners. Each learner gets different training set of *n* instances extracted independently with replacement from the learning set. The bootstrap replication of training instances is not the only source of randomness. In each node of the tree the splitting attribute is selected from a randomly chosen sample of attributes. As the training sets of individual trees are formed by bootstrap replication, there exists on average $\frac{1}{e} \approx 36.8\%$ of instances not taking part in construction of the tree [17]. The random forest performs well compared to some other popular classifiers. Also, it has only two parameters to adjust: (i) the number of variables in the random subset at each node, and (ii) the number of trees in the forest. It learns fast.

IV. Proposed Method

This paper explores the utilization of an error correcting code and random forests learner for multilabel classification. The proposed method is called MultiLabel Bose, Ray-Chaudhuri, Hocquenghem Random Forests (ML-BCHRF). The block diagram description of the ML-BCHRF is shown in Fig. 2.

The method first transforms the set of labels *L* using the Bose, Ray-Chaudhuri, Hocquenghem (BCH) encoding algorithm. For a *k* class dataset, each set of labels that is associated with an instance containing *k* binary values is treated as a message codeword and is transformed into an *n* bit binary values where *n* > *k*. the n bit binary word is called the encoded message. Then, the multilable classification problem is decomposed into *n* binary classification problems. Next, *n* random forests classifiers are developed one for each binary class. After that, the *n* classification decisions of *n* binary classifiers are transformed using the BCH decoding algorithm and again *k* binary values are obtained. Therefore, the advantage of the error-correcting properties of the BCH code is incorporated into the system that helps correct possible misclassification of some individual *n* binary classifiers.

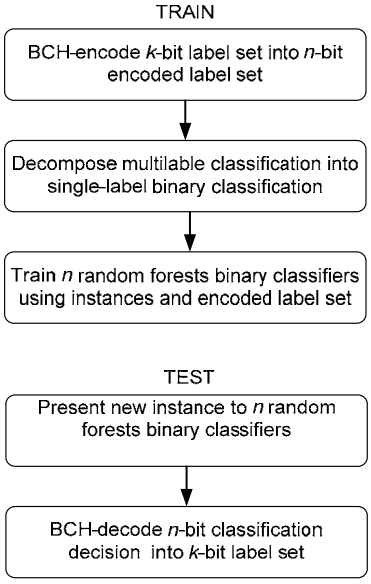

Figure 2. Block diagram description of the proposed ML-BCHRF method.

For classification of a new instance, its features are independently presented to *n* binary classifiers. Then the *n* classification decisions of *n* binary classifiers are transformed into *k* binary values using the BCH decoding algorithm. The error-correcting is applied during this transformation that helps correct possible misclassification of some individual *n* binary classifiers. The bits of the *k* resultant binary values that are '1' indicate that the instance belong to the associated class.

V. Experimental Results

To evaluate ML-BCHRF, its performance was evaluated against a number of exiting methods on three different datasets. These are among the popular benchmark datasets for multi-label classification. Their characteristics are presented in Table II.

The ML-BCHRF employs the random forest learner as its base classifier. The random forest learner has two important parameters, called number-of-trees-to-grow and number-of-variables-at-each-split, that can be varied to get the best number of tree within the forest for the specific training data.

TABLE II.
Characteristics of Benchmark Datasets

Dataset	Features	Classes	Train	Testing
Scene	294	6	1211	1196
Yeast	103	14	1500	917
Mediamill	120	101	30993	12914

In the first experiment, we trained and tested the ML-BCHRF on the Yeast dataset. The (63, 16) BCH encoder was used and two dummy bits were added to the label set making

© 2009 ACADEMY PUBLISHER

it have 16 binary bits. We used number-of-trees-to-grow = 60 and number-of-variables-at-each-split = 20. However, these two parameters can be varied to achieve better results. The ML-BCHRF results were compared against those of MMP [6], AdaBoost.HM [7], ADTBoost.HM [8], LP [5], BR [2], RankSVM [9], ML-KNN [3], RAKEL [5], 1vsAll SVM [10], and ML-PC [11] found in the literature. Table III shows the experimental results for the Yeast dataset.

TABLE III.
RESULTS FOR YEAST

Method	Hamming Loss
MMP	0.297
ADTBoost.HM	0.215
AdaBoost.HM	0.210
LP	0.202
BR	0.199
RankSVM	0.196
ML-KNN	0.195
RAKEL	0.193
1vsAll SVM	0.191
ML-PC	0.189
ML-BCHRF	0.188

In the second experiment, we trained and tested the ML-BCHRF on the Scene dataset. The (31, 6) BCH encoder was used. We used number-of-trees-to-grow = 60 and number-of-variables-at-each-split = 20. The ML-BCHRF results were compared against those of BR [2], LP [5], RAKEL [5] found in the literature. Table IV shows the experimental results for the Scene dataset.

In the third experiment, we trained and tested the ML-BCHRF on the Mediamill dataset. The (255, 107) BCH encoder was used and six dummy bits were added to the label set making it have 107 binary bits. We used number-of-trees-to-grow = 60 and number-of-variables-at-each-split = 20. The ML-BCHRF results were compared against those of LP [5], BR [2], ML-KNN [3], RAKEL [5] found in the literature. Table V shows the experimental results for the Mediamill dataset.

TABLE IV.
RESULTS FOR SCENE

Method	Hamming Loss
BR	0.114
LP	0.099
RAKEL	0.095
ML-BCHRF	0.074

The experimental results show that ML-BCHRF has performed better than its reported counterparts. It has performed very well for three datasets of varying characteristics. The reason for the demonstrated performance relates to the mixture of: (i) the error correcting capability of the BCH code, and (ii) the superior performance of the random forest learner.

TABLE V.
RESULTS FOR MEDIAMILL

Method	Hamming Loss
LP	0.046
BR	0.038
ML-KNN	0.031
RAKEL	0.030
ML-BCHRF	0.028

VI. CONCLUSION

A method was proposed that BCH-encodes labels and then decomposes the problem into binary classification. One random forests classifier is developed for each binary class. The classification decisions are BCH-decoded using the BCH decoding algorithm and again k binary values are obtained. The experimental results show that the proposed method has performed better than its reported counterparts. Future work will include experiments in which the parameters of the random forests learner could be varied for achieving better results.

REFERENCES

[1] G. Tsoumakas and I. Katakis, “Multi-label classification: An overview,” *International Journal of Data Warehousing and Mining*, vol. 3, no. 3, pp. 1-13, 2007.
[2] K. Brinker, J. Furnkranz, and E. Hullermeier, “A united model for multilabel classification and ranking,” *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI '06)*, Riva del Garda, Italy, 2006, pp. 489-493.
[3] M.-L. Zhang and Z.-H. Zhou, “A k-nearest neighbour based algorithm for multi-label classification,” *Proceedings of IEEE GrC'05*, Beijing, China, 2005, pp. 718-721.
[4] M.-L. Zhang and Z.-H. Zhou, “Multi-label neural networks with applications to functional genomics and text categorization,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 10, pp. 1338–1351, 2006,
[5] G. Tsoumakas and I. Vlahavas, “Random k-labelsets: An ensemble method for multilabel classification,” *Proceedings of the 18th European Conference on Machine Learning (ECML 2007)*, Warsaw, Poland, 2007, pp. 406-417.
[6] C. Crammer and Y. Singer, “A family of additive online algorithms for category ranking”, *Machine Learning Research*, 3, pp. 1025-1058, 2003.
[7] R. E. Schapire and Y. Singer, “BoosTexter: A boostingbased system for text categorization”, *Machine Learning*, 39, pp. 135-168, 2000.
[8] F.D. Comite, R. Gilleron and M. Tommasi, “Learning multi-label alternating decision tree from texts and data”, *Proceedings of MLDM 2003, Lecture Notes in Computer Science*, 2734, Berlin, 2003, pp. 35–49.
[9] A. Elisseeff and J. Weston, “A kernel method for multilabelled classification”, *Proceedings of NIPS’02*, Cambridge, 2002.
[10] J. Platt, “Probabilistic Outputs for Support Vector Machines and Comparison to Regularized Likelihood Methods”. *Adv. in Large Margin Classifiers*. MIT Press, pp. 61–74. 1999.
[11] M. Petrovskiy, “Paired comparisons method for solving multi-label learning problem,” *Proceedings of the Sixth International Conference on Hybrid Intelligent Systems (HIS'06)*, 2006.
[12] C.-H. Wei, “Channel coding notes,” [Online]. Available: http://cwww.ee.nctu.edu.tw/course/channel_coding/
[13] A. Rudra, “BCH codes,” Lecture 24, Error Correcting Codes: Combinatorics, Algorithms and Applications (Spring 2009), 2009.
[14] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, pp. 5-32, 2001.
[15] A. Liaw and M. Wiener, "Classification and regression by randomForest," *R News*, vol. 2, pp. 18-20, 2002.
[16] Y. Freund and R. E. Schapire, "A short introduction to boosting," *Journal of Japanese Society for Artificial Intelligence*, vol. 14, pp. 771--780, 1999.
[17] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, 1996.

© 2009 ACADEMY PUBLISHER