Abstract—This paper describes how a single multi-output MLP can be applied to multi-label classification tasks, and reports on the application of the technique to predicting gene function for arabidopsis—a small flowering plant, and one of the most completely sequenced eukaryotic genomes. Comparison of the classification characteristics of the multi-output MLP with that of multiple binary classifiers reveals several differences, most notably a more rapid fall-off in sensitivity as the output cutoff value is increased. These differences are due to an increased peakedness in the distribution of output values as compared with the distribution of outputs from binary networks. Various explanations are offered to account for this.

I. INTRODUCTION

In most classification problems, the examples to be classified belong to one of a set of mutually exclusive classes. A classifier is trained using a collection of training examples, and the classifier is then used to classify a new example into the single class to which it is most likely to belong. If the number of mutually exclusive classes is two, the classifier is a binary classifier, otherwise it is a multi-class classifier.

However, there also exist problem domains in which examples may belong to more than one class simultaneously. A simple example is music genre classification, in which a song may be described simultaneously as both blues and rock. Other examples include document classification, scene classification, and medical diagnosis. Importantly, in these domains, an example’s membership is not shared amongst the classes to which it belongs; rather, the example belongs fully to each of those classes, i.e., the song is a blues song AND a rock song. Classification problems in which examples may belong simultaneously to more than one class are usually referred to as multi-label classification problems. In order to avoid possible confusion, in this paper we will use the term single-label multi-class to refer to multi-class problems in which examples may belong to only one class.

Initial work in the area of multi-label classification concentrated on applications to text categorization, in which a document may belong simultaneously to several topics. McCallum [1] used a generative approach in which a mixture model is assumed to generate the documents, and Expectation-Maximization (EM) [2] is utilized to learn the mixture weights and word distributions in each of the components. Generative models were also used by Ueda and Saito [3]. Schapire and Singer [4] developed a boosting-based approach for the text categorization problem, and this was subsequently used by Comte et al [5]. Other multi-label classification domains have included scene classification [6], and prediction of gene functional class [7, 8]. Examples involving applications of neural networks to multi-label classification problems include Boutell et al [6] and Goncalves and Quaresma [9], who apply Support Vector Machines (SVMs) to the problems of semantic scene classification and document classification respectively.

SVMs are binary classifiers, and, as such, can only be used to distinguish between two classes. The usual way of applying SVMs to multi-label (and also single-label multi-class) classification problems is to train a separate SVM for each class, with examples not belonging to the class used as negative examples for that class. Thus, assuming that $S$ is the set of classes to which some example belongs, the example is treated as a positive example when training the classifier for class $s \in S$, and negative when training the classifier for class $s \notin S$. A novel example can then be classified by applying its attribute vector as input to each of the binary classifiers, and attaching class labels accordingly. One criticism of the multiple binary classifier approach is that it is cumbersome, as a separate binary classifier must be trained and stored for each class. A second, and more important criticism that has been directed at this approach is that it does not exploit the dependencies between the labels [1, 8, 10, 11].

An alternative to using multiple binary classifiers is to use a single neural network with multiple outputs (one output node per class), and an $m$-of-$n$ target vector encoding in which the target vector has length $n$ (where $n$ is the number of classes), and consists of $m$ + $1$s (corresponding to the $m$ multiple classes to which the example belongs) and $n$–$m$ 0s. While this approach appears natural and rather obvious, and has been proposed as a possible extension in work reported by Boutell et al [6], rather surprisingly, to the present authors’ knowledge there has been no report on the application of this approach to the multi-label classification problem.

In this paper we describe the application of this approach to the problem of predicting gene function for arabidopsis—a small flowering plant, and probably the most completely sequenced eukaryotic genome. The main purpose of the research was to determine whether there is any significant difference in classification performance between using multiple binary classifiers and a single multi-class classifier.

The paper is structured as follows. Section II addresses important considerations in the generic application of neural networks to the multi-label classification problem. These include network architecture and choice of activation function, assembly of training examples, and evaluation of classifier performance. Section III describes the gene
function prediction domain. Section IV describes experiments and presents results. Section V is a discussion, and Section VI concludes the paper.

II. MULTI-LABEL CLASSIFICATION USING MLPs

In the following, we assume a multi-layer perceptron (MLP) consisting of an input layer containing a unit for each attribute, a single layer of hidden units, and an output layer containing a node for each class in the domain. We first consider the appropriate activation to use at the output nodes.

A. Output Activations

Almost invariably, the output unit activation function used in MLPs is either the linear, logistic, or softmax function. The choice is justified on maximum likelihood considerations. For regression problems (i.e., problems in which target values are continuous values) the output activation should be linear; for binary classification problems (i.e., problems in which examples must be classified as belonging to one of two mutually exclusive classes) in which a single output unit is used the appropriate activation function is the logistic function; and for single-label multi-class classification problems (i.e., problems involving more than two classes), the usual choice is the softmax function,

\[ y_k = \frac{\exp(a_k)}{\sum_k \exp(a_k)}, \]

where \( a_k \) is the weighted sum (prior to any thresholding) at output \( k \). The motivation for using this activation function is that it has the two properties \( 0 \leq y_k \leq 1 \) and \( \sum_k y_k = 1 \), allowing the value at a particular output node to be interpreted as the probability, conditional on the training data, that the input example belongs to the class corresponding to that output node. Note that the second of the two properties implicitly assumes that examples can belong to one class only; i.e., that classes are mutually exclusive.

In the multi-label problem, classes are NOT mutually exclusive: by definition, an example may belong to more than one class. Thus, the softmax function is clearly inappropriate for these problems. Instead, we replace the output activation function at each output with the logistic function

\[ y_k = \frac{1}{1 + \exp(-a_k)}, \]

which has the property \( 0 \leq y_k \leq 1 \), but does not constrain the sum over all outputs to be unity. Thus, the output at a particular node now represents the posterior probability of belonging to the class corresponding to that node, without the assumption that the classes are mutually exclusive.

B. Target Vector Encoding

In single-label multi-class classification problems, a 1-of-\( n \) encoding is usually used; i.e., the target vectors consist of a binary vector of length \( n \), where \( n \) is the number of classes. A target vector contains a 1 in the position corresponding to the target class, and 0’s elsewhere. For the multi-label multi-class problem, we propose an \( m \)-of-\( n \) target vector encoding consisting of \( m \) 1s (corresponding to the \( m \) multiple classes to which the example belongs) and \( n-m \) 0s.

C. Determining Classifier Output

In single-class multi-label classification problems, a novel example is classified as belonging to the (single) class for which the output at the corresponding network node is a maximum. In the multi-label case some other decision criteria must be used, as examples may belong to more than one class. Given that the output values of the network represent posterior probabilities of class membership, the obvious approach is to use thresholding, in which an example with attribute vector \( x \) is classified as belonging to a set of classes \( C \) if \( P(c | x) > T, \forall c \in C \), where \( T \) is a threshold value that must be determined. However, because network outputs represent the conditional probability of class membership, and because the prior probabilities of membership to the various classes may differ widely, we propose using a separate threshold for each class; i.e., classify \( x \) as belonging to class \( c_i \) only if \( P(c_i | x) > T_i \), where \( T_i \) is the threshold for class \( c_i \). A reasonable choice would be to set each \( T_i \) equal to \( P(c_i) \) (i.e., the prior probability of membership to \( c_i \)) or to some multiple, \( k \), of this prior, where \( k \) is constant over all classes, \( c_i \). Thus, \( x \) is classified as belonging to \( c_i \) only if \( P(c_i | x) > kP(c_i) \).

D. Classifier Evaluation

In contrast to single-class classification, in which a class prediction is either correct or incorrect, predictions in multi-label multi-class problems may be partially correct: some, but not all, of the classes to which a test example belongs may not have been predicted; or, in addition to it being correctly predicted as belonging to some classes, an example is also predicted as belonging to additional classes.

Various measures have been proposed for evaluating multi-label multi-class classification results. For example, Schapire and Singer [4] use one-error, which evaluates the frequency with which the top-ranked class does not appear in the actual classes; coverage, which measures how far one must proceed, on average, down the list of predicted classes in order to cover all of the actual classes to which the examples belong; and average precision, which is a modified version of the precision measure often used in IR systems. Additional evaluation measures have also been proposed in Boutell et al [6].

While these measures are appropriate in many cases, they may prove unreliable in cases in which a significant number of class labels are missing. For example, in the problem of predicting gene function annotation, it is highly likely that many genes have functions which are currently unknown. This means that in general, we could expect a relatively high number of false positives (i.e., actual negatives predicted as positives). On such domains, we feel it more appropriate to use more conventional classification performance measures,
but to take into account the relative balance between false positives and false negatives by measuring sensitivity and predictive value (defined in Section IV).

III. PREDICTING GENE FUNCTION

The data used in this paper comes from microarray samples from the Arabidopsis thaliana species. Arabidopsis is a small flowering plant related to cabbage and mustard. It has been widely studied because it is used as a model organism in plant biology. It was chosen as such because it has a relatively small genome for a plant species, with only 5 chromosomes, and is simple and quick to grow. Its genome has been fully sequenced, and there are many existing microarray samples for it.

A microarray is a technique to determine the gene expression levels; that is, how much of a gene is expressed in a cell at a particular point in time. It is useful for comparisons within and between species, with the latter due to the fact that many species share the same genes. A single microarray consists of a large array of sensors, each corresponding to a different gene. After processing, each sensor shows the amount that a particular gene is being expressed.

In general, it is relatively difficult to produce a large number of microarray samples, especially in relation to the number of genes measured by each sample. In our case, because Arabidopsis has been broadly studied, we do have a relatively large number of samples. These samples come from a wide range of situations; from different locations on the same plant, from different times during a plant's life, and from plants undergoing different environmental stresses. In this way, the samples give a broad overview of Arabidopsis, showing how different parts of it function under a wide variety of environmental conditions.

The function of many of the microarray identified genes has been classified into functional categories from a standard gene ontology [12]. The evidence source for this functionality comes from a number of sources, such as direct experimental evidence; for example, by inhibiting a gene and observing the resulting organism. A weaker type of evidence is called synteny, in which a gene's function is inferred by looking at the function of the same gene in a different species.

The gene ontology describes gene functionality at several levels. At one extreme are the three gross categories: Molecular Function, Biological Process, and Cellular Component. At the other extreme are the GO terms, which are the finest level of categorization. In our version of the gene ontology, there were 2406 such categories; for example microsporogenesis. While it would be useful to be able to predict such fine grained function, the use of these categories alone leads to too few examples in each category. This problem has been approached in two ways. One solution is to use the hierarchical nature of the ontology, merging small categories with their siblings, rolling up to the parent class [13]. Another approach is to use an alternate categorization, parallel to the GO terms, called GOSLIM. This is a less detailed categorization of the genes, with only 46 different categories.

Biological systems are complex. A single gene can produce various proteins, which can each be used in various biological processes. In this way, a gene can correctly belong to several functional categories. Our research aims for accurate prediction of all of these categories for a single gene.

IV. EXPERIMENTAL RESULTS

This section describes the application of MLPs to the prediction of arabidopsis functional annotation.

A. Data Preparation

As described above, in our experiments we used the GOSLIM function annotations, which consist of 46 categories. Of these, 14 are of type 'unknown' or 'other' (e.g., other cellular processes, other metabolic processes, etc.); these we excluded from our experiments, leaving 32 functional annotations. After removing these, the number of genes which had at least one annotation was 10,003, with the average number of annotations per gene being 2.64. The number of attributes used for predicting functional annotation is 1876. (The original dataset had 1877, but one attribute had approximately half the values missing, so we deleted that attribute). In order to reduce the dimensionality of the data, we performed a principal components analysis on the attribute values and retained the first 10 principal components (these account for more than 99% of the variability). We then divided the 10,003 training examples into two approximately equal-sized sets—one to be used for training, and one to be used for testing.

B. Single Network with Multiple Outputs

In a first set of experiments, we applied the technique described in Section II; i.e., a single network with multiple outputs. In order to gain some insight into the degree of overfitting (if any) that might occur during training, we experimented using networks with various numbers of hidden units and weight regularization coefficient values. Networks were trained using the Conjugate Gradients algorithm, and the error on the test set was monitored during training. We observed that it was only when the number of hidden units exceeded approximately 10, and in which the value of the weight regularization coefficient was less than about 0.01 that any overfitting was observed; however, the degree of overfitting was extremely slight in this case, so these parameters were used in generating the results described below. 1

In order to test the ability of the network to predict the annotations of test examples, suitable thresholds must be determined for each of the 32 outputs. In general, small thresholds will result in a large number of false positives (i.e., examples which do not have an annotation, but are

1 Strictly speaking, this type of validation should usually be performed using examples other than those to be used for testing. However, the purpose here was to obtain only a rough idea of how many hidden units, and what regularization coefficient value is appropriate for this data. Moreover, several training/test set partitions of the 10,003 examples were used to determine this. Thus, there is no danger that the results presented herein are affected by this procedure.
assigned the annotation by the network), while large thresholds will result in a large number of false negatives (i.e., examples which have an annotation, but are not assigned the annotation by the network). As described in Section II, we select a separate threshold for each network output. A test example is assigned an annotation if the network output corresponding to that class exceeds $kP(c_i)$, where $P(c_i)$ is the prior probability of an example having annotation $c_i$ (determined by dividing the number of examples having an annotation by the total number of examples), and $k$ is a constant which we refer to as the threshold scaling factor.

We follow the convention used in many medical diagnosis tasks, whereby classification performance is measured in terms of sensitivity, predictive value, and overall accuracy. These are defined as follows:

\[
\text{sensitivity} = \frac{TP}{TP+FN} \\
\text{predictive value positive} = \frac{TP}{TP+FP} \\
\text{accuracy} = \frac{TP+TN}{TP+FP+TN+FN}
\]

where $TP = \text{number of true positives}$, $FP = \text{number of false positives}$, $TN = \text{number of true negatives}$, and $FN = \text{number of false negatives}$.

Table I shows the sensitivity, predictive value positive, and accuracy of the network predictions on the test set examples, corresponding to a threshold scaling factor of $k = 0.75$. Also shown is the count of the number of genes possessing the respective annotation.

A $k$ value of 0.75 was deliberately selected so as to achieve a high sensitivity. By adjusting this threshold, a difference balance can be achieved between sensitivity, predictive value, and overall accuracy. Figure 1 shows how classification performance on test examples, averaged over all 32 annotation classes, varies with the value of $k$. Low thresholds, the sensitivity is high, but the overall accuracy is low due to the large number of false positives. Conversely, large thresholds result in low sensitivity, but the overall accuracy is high. This is due to the fact that most examples do not have an annotation, and that as the threshold is increased, fewer annotations are predicted. Note that the predictive value positive has a maximum of approximately 10%, corresponding to $k = 0.95$. While this may appear alarmingly low, it simply reflects the fact that many genes have functions which are as yet unknown, which is almost certainly the case. This is supported by Fig. 2, which shows classification performance on the training examples, and is remarkably similar to Figure 1. In other words, there are probably a very significant number of functional annotations missing from both the training data and the test data.

![Fig. 1. Test set classification performance as a function of threshold scaling factor, $k$. Low thresholds result in high sensitivity, but low overall accuracy due to the large number of false positives; large thresholds result in low sensitivity due to the large number of false negatives.](image_url)
C. Multiple Binary Classifier Networks

We would like to know how the use of a single neural network compares with the use of multiple (binary classifier) networks. To this end, we conducted the same experiments, but this time by training a separate binary classifier neural network for each of the 32 annotation classes. The number of hidden layer units was again 10, and thresholds were determined the same way as previously. Table II compares the classification performance of the single network (i.e., the network used previously), with that of multiple binary network classifiers on both training and test data ($k = 0.75$).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single Network</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train</td>
<td>0.912</td>
<td>0.094</td>
<td>0.252</td>
</tr>
<tr>
<td>Test</td>
<td>0.917</td>
<td>0.093</td>
<td>0.251</td>
</tr>
<tr>
<td><strong>Multiple Binary Networks</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train</td>
<td>0.893</td>
<td>0.102</td>
<td>0.342</td>
</tr>
<tr>
<td>Test</td>
<td>0.849</td>
<td>0.099</td>
<td>0.340</td>
</tr>
</tbody>
</table>

Note that sensitivity for the multiple binary networks is significantly lower than that for the single network (0.849 versus 0.917 on test examples); however, overall accuracy for the single network is less than that for the multiple binary networks (0.251 versus 0.340). To determine whether this is a general trend that occurs across other threshold values, we ran further experiments, $k$. The results on both test and training examples are shown in Figure 3, and it can be clearly seen that the observation described above holds for all values of $k$ less than approximately 1.0. However, as $k$ increases beyond 1.0, sensitivity falls off more rapidly in the case of the multi-output network.

V. DISCUSSION

It is tempting to try to explain the different classification performance characteristics of the two approaches as a result of overfitting in the case of multiple binary networks. Indeed, the multiple binary networks would be expected to be more prone to overfitting on account of the greater number of weights per output class. That is, each binary network contains 121 weights, while in contrast the multi-class network contains a total of 462 weights, which averages to only 14.4 per class label. However, if overfitting were occurring, we would expect to see a difference between classification characteristics on training and test examples. As there is no such significant difference, we rule out the possibility that the phenomenon is due to overfitting.

The use of the logistic activation function at the outputs of the MLP ensures that the outputs approximate the posterior probability that a gene contains a function annotation. This is the case for both the multi-output network, as well as for multiple binary networks. However, the different classification characteristics of the two approaches implies that the probabilities are being modeled differently in each case. To illustrate this, consider once again Fig 3, and note that the classification characteristics of multiple binary networks can be made to more closely resemble those of the
multi-output network simply by adjusting the value of \( k \). For example, for \( k \) values less than 1, lowering the value of \( k \) will increase the sensitivity and decrease overall accuracy. This suggests that in the lower probability ranges the output values of the binary classifiers are on average lower than those for the multi-class network. Conversely, for \( k \) values greater than 1, the value of \( k \) would need to be increased in order for the binary network classification characteristics to be brought into line with those of the multi-output network. This suggests that in the upper probability ranges the output values of the binary classifiers tend to be higher, on average, than those for the multi-class network. Thus, it appears that the distribution of outputs for the binary networks tend to be less peaked than those for the multi-output network.

According to [14], accurate approximation of outputs as posterior probabilities relies on a number of factors: (1) the network must be large enough, and a global minimum must be found; (2) the amount of training data should be infinite, and (3) the a priori class probabilities of the test set must be correctly represented in the training set. Given that we have used the same training and test sets in each case, the different performance characteristics are unlikely to be due to factor (3) above. Rather, we feel that the differences are a result of a combination of factors (1) and (2). While a single binary network has fewer weights than the multi-output network, the multi-output network has far fewer weights per label than a single binary network (14.4 and 121 respectively). Under the presumption that larger networks require a greater amount of training data than do smaller networks, the binary networks have relatively less training data than the multi-output network. Thus, the approximation of the outputs as posterior probabilities is not expected to be as accurate for multiple binary networks as it is for the single multi-output network. We can infer from this that the outputs of the multi-output network are better estimates of the posterior probabilities of class membership than are the outputs of the binary networks.

As stated in the introduction, one of the criticisms of the multiple binary classifier approach is that it does not exploit dependencies that may exist between class labels. This is because a separate classifier is trained for each class label, and the class labels are thus treated as being independent. The situation is not entirely different in the case of the multi-output network, in which case the hidden-to-output layer weights are updated in exactly the same way as per binary networks. The difference, however, is that in the multi-network case, all class labels are forced to share the same set of input-to-hidden layer weights. If there is any difference between the two approaches in regards to the ability to exploit dependencies between class labels, these differences must be implicit in the fact that input-to-hidden layer weights are shared in the case of the multi-output network. The above considerations may also help explain the greater peakedness in the distribution of outputs for the multi-output network as compared with binary networks. The relatively greater number of free parameters (i.e., weights) used in binary networks, or, equivalently, the effectively smaller amount of training data relative to number of weights, may allow the binary networks to achieve their classification accuracies using a larger range in output values than does the single multi-output network, which must model probabilities on an effectively larger amount of training data.

VI. CONCLUSION

A technique for applying a single multi-output MLP to multi-label classification problems has been described. When applied to the task of predicting gene function from microarray data, the use of this technique has been found to yield different classification characteristics to the more standard approach of using multiple binary classifiers. These differences appear to be due to a greater peakedness in the distribution of outputs for the multi-output network in comparison to those of binary networks. This increased peakedness is most likely due to fact that the single multi-output network contains fewer weights per output label than does a single binary network. Effectively, this means that the multi-output MLP uses a higher number of training examples per class, resulting in a more accurate approximation of outputs as posterior probabilities. In future work we intend to compare the performance of these (discriminative) approaches with that of generative approaches. We also intend to apply these approaches to a number of other domains such as document classification and image scene classification.

ACKNOWLEDGMENT

We would like to thank David Edwards, Ross Chapman, Christopher Love and Tim Erwin for useful discussions and help with the preparation of data.

REFERENCES


