BIOINFORMATICS

Vol. 17 Suppl. 1 2001
Pages S316–S322

Molecular classification of multiple tumor types

Chen-Hsiang Yeang, Sridhar Ramaswamy, Pablo Tamayo, Sayan Mukherjee, Ryan M. Rifkin, Michael Angelo, Michael Reich, Eric Lander, Jill Mesirov and Todd Golub

Center for Genome Research, MIT Whitehead Institute, One Kendall Square, Cambridge, MA 02139, USA

Received on February 5, 2001; revised and accepted on April 2, 2001

ABSTRACT

Using gene expression data to classify tumor types is a very promising tool in cancer diagnosis. Previous works show several pairs of tumor types can be successfully distinguished by their gene expression patterns (Golub *et al.* (1999), Ben-Dor *et al.* (2000), Alizadeh *et al.* (2000)). However, the simultaneous classification across a heterogeneous set of tumor types has not been well studied yet. We obtained 190 samples from 14 tumor classes and generated a combined expression dataset containing 16063 genes for each of those samples. We performed multi-class classification by combining the outputs of binary classifiers. Three binary classifiers (k-nearest neighbors, weighted voting, and support vector machines) were applied in conjunction with three combination scenarios (one-vs-all, all-pairs, hierarchical partitioning). We achieved the best cross validation error rate of 18.75% and the best test error rate of 21.74% by using the one-vs-all support vector machine algorithm. The results demonstrate the feasibility of performing clinically useful classification from samples of multiple tumor types.

Contact: chyeang@mit.edu

INTRODUCTION

Modern cancer treatment is based on the accurate determination of a tumor's site of origin. In general, pathologists utilize a variety of microscopic, genetic, and immunologic techniques to make site-specific diagnosis. However, current techniques are limited in their ability to distinguish different tumor types. Many specimens are incorrectly classified due to their morphological similarity to other tumor types. Moreover, a large number of samples remain poorly differentiated and difficult to relate to any known tumor types including their sites of origin.

The emerging technology of gene expression analysis is a promising tool for cancer diagnosis. In principle, tumor gene expression profiles might serve as molecular fingerprints that would allow for the accurate classification of tumors. The underlying assumption is that samples from the same tumor class share some expression profile patterns unique to their class. In addition, these molecular fingerprints might reveal newer taxonomies that previously have not readily been appreciated.

Previous works have demonstrated successful distinction of several tumor types from expression profiles. For example, leukemias arising from different precursors (Golub *et al.* (1999)) and B-cell lymphoma (Alizadeh *et al.* (2000)). Low error rates (0-10%) are achieved in these morphological and lineage classifications. However, these works only tackle binary classification problems. As there are over a hundred types of cancer (Hanahan & Weinberg (2000)) and potentially even more subtypes, for any practical applications it is essential to develop multiclass methodologies for molecular classification. By extending to simultaneous classifications of multiple tumor types, the problem becomes intrinsically more difficult. As there is only one positive class (the sample's true class) but many negative classes (all other classes), misclassifications are more likely to occur due to the noise or variations of the expression data. The resources and time required for collecting and genotyping specimens limit the number of samples in each class, which makes statistical inference very difficult.

In this paper we tackle the problem of cancer classification in the context of multiple tumor types. We applied the output coding scheme (Dietterich & Bakiri (1995), Allwein *et al.* (2000)) which combines binary classifiers to perform multiclass prediction. Section 2 describes the binary classifiers and combination methods we used. Section 3 describes the classification outcome on tumor dataset and discusses the results. Section 4 draws conclusions and points out the current and future directions with regard to this project.

MULTICLASS CLASSIFICATION

Tumor type classification can be posed as a multiclass prediction problem in machine learning. Let $S = \{(\mathbf{x_1}, y_1), ..., (\mathbf{x_m}, y_m)\}$ be a set of m training samples where the $\mathbf{x_i} \in \mathcal{X}$ is the input of the ith sample (in this case, the expression profile of 16063 genes) and

$y_i \in \mathcal{Y} = \{1, 2, ..., k\}$ is its multiclass label (in this case, the tumor class). A multiclass classifier is a function $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$ which maps an instance $\mathbf{x}$ into a label $\mathcal{F}(\mathbf{x})$. Two approaches are commonly used to solve this problem. One approach is to generate the function $\mathcal{F}$ directly, for example, decision trees (Breiman (1984)), naive Bayes algorithm (Duda (2001)) or k-nearest neighbors (Duda (2001)). The other approach is to construct $\mathcal{F}$ by combining a number of binary classifiers. Many binary classifiers such as weighted voting (Golub *et al.* (1999)), support vector machines (Vapnik (1998)), and multi-layer perceptrons (Minsky (1969)), are difficult to extend into multiclass versions directly. Inspired by the satisfactory performance of binary classifiers in tumor classification we decided to adopt the second approach.

Among the 16063 genes, usually only a small portion of them are related to the distinction of tumor classes. Therefore, we want to choose a subset of genes based on the strength of their correlation to separate tumor classes. This procedure is called feature selection (here we define each gene as a feature). We used the feature selection algorithm which was developed in our previous work (Golub *et al.* (1999)). Let c be a (binary) class vector and g be the expression vector of a gene over m training samples. Let μ_1 and μ_2 be the means of g within the samples of classes 1 and 2, and σ_1 and σ_2 be the standard deviations of g within the samples of classes 1 and 2. We then define $P(g, c) = \frac{(\mu_1 - \mu_2)}{(\sigma_1 + \sigma_2)}$ to be the signal-to-noise ratio of g with respect to class vector c. This quantity captures the separation between two classes and the variation within individual classes. Figure 1 illustrates the signal-to-noise ratio for binary class prediction. An "ideal" marker gene expression profile is a binary vector which is 1 among all the samples in class A and 0 among all the samples in class B (or vice versa). This profile does not happen since the reading of microarray experiments is continuous. We are looking for "good" marker genes which are close to the binary expression profile. The signal-to-noise ratio measures how well the expression profile of a real gene approximates the ideal marker gene profile. The top genes in terms of the absolute values of the signal-to-noise ratio are chosen to build binary classifiers.

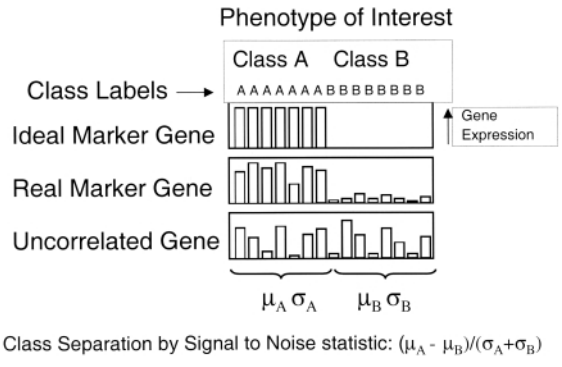

Fig. 1. Illustration of the signal-to-noise metric.

Binary classifiers

The weighted voting (WV) algorithm directly applies the signal-to-noise ratio to perform binary classification. For a chosen feature of a test sample it measures its distance with respect to the decision boundary $b = \frac{1}{2}(\mu_1 + \mu_2)$, which is located halfway between the average expression levels of two classes. If the value of this feature falls on one side of the boundary, a vote is added to the corresponding class. The vote $V = P(g, c)(x - b)$ is weighted by the distance between the feature value and the decision boundary and the signal-to-noise ratio of this feature determined by the training set. The vote for each class is computed by summing up the weighted votes made by selected features for this class. The weighted-voting algorithm with the signal-to-noise ratio metric is very similar to Bayesian binary detection under the assumption of a Gaussian distribution within each class (Slonim *et al.* (2000)). In Bayesian detection the discriminant function is $V = (\frac{\mu_1 - \mu_2}{\sigma^2})(x - b)$ assuming that the within-class variances σ^2 of classes 1 and 2 are identical.

We are interested not only in predicting class labels but also in the confidence of the prediction. To fulfill this requirement the output of the binary classifier is a real number instead of a binary value. The sign of the output denotes the class label, whereas its absolute value denotes the confidence of prediction. In the weighted voting algorithm the confidence value is $c = \frac{V_{win}}{(V_{win} + V_{lose})}$, where V_{win} is the overall vote acquired by the winnning class and V_{lose} is the overall vote acquired by the losing class.

The k-nearest neighbors (kNN) algorithm is a simple but effective classification algorithm. It is widely used in machine learning and has numerous variations (Duda (2001)). Given a test sample of unknown label, it finds the k nearest neighbors in the training set and assigns the label of the test sample according to the labels of those neighbors. The vote from each neighbor is weighted by its rank in terms of the distance to the test sample: the nearest neighbor's vote is multiplied by 1, the second nearest neighbor's vote is multiplied by $\frac{1}{2}$, and so on. This weighting scheme requires at least 3 consensus votes against the nearest neighbor in order to overturn the decision drawn from it.

The confidence values of the kNN algorithm are related to the relative strength of the weighted votes for each class. If the vote for class 1 dominates the vote for class 2 (e.g. all the k nearest neighbors belong to class 1),

then the confidence of choosing class 1 as the predicted label is high. We use both the normalized vote for the winning class ($\frac{V_{win}}{(V_{win}+V_{lose})}$) and the negative entropy of the normalized votes ($1 - H(\frac{V_{win}}{(V_{win}+V_{lose})}, \frac{V_{lose}}{(V_{win}+V_{lose})})$) as confidence values, but only report the results based on the negative entropy confidence.

The support vector machine (SVM, Vapnik (1998)) is one of the most powerful supervised learning algorithms. It finds a hyperplane $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + \mathrm{b}$ which separates positive and negative training samples and maximizes the margin between the samples and the hyperplane (recall $\mathbf{x}$ is the input data and y is the class label). This task can be formulated as an optimization problem: minimize $\|\mathbf{w}\|^2$ subject to the constraints $y_i(\mathbf{x_i} \cdot \mathbf{w} + \mathrm{b}) - 1 \geq 0 \forall i$. The discriminant function can be written as the following form: $f(\mathbf{x}) = \sum_i w_i y_i (\mathbf{x} \cdot \mathbf{x_i}) + b$, where w_i's and b can be obtained by solving the quadratic programming problem. An important characteristic is that only the training samples which are misclassified or lie between the hyperplanes $(\mathbf{x} \cdot \mathbf{w} + \mathrm{b}) = +1$ and $(\mathbf{x} \cdot \mathbf{w} + \mathrm{b}) = -1$ contribute to the non-zero terms w_i. These samples are called the "support vectors" of the predictor. Only the support vectors are relevant in classification.

The output of the predictor $f(\mathbf{x})$ can be used as a confidence value. It is the margin between the sample and the decision hyperplane (normalized by $\frac{1}{\|w\|}$). A loss function is a function of margin values. A commonly used loss function in SVM is the hinge loss: $L(\mathbf{x}, \mathrm{y}) = (1 - y(\mathbf{w} \cdot \mathbf{x}) + \mathrm{b})_+$, where $(z)_+ = max\{z, 0\}$. When the sample falls out of the region $y(\mathbf{x} \cdot \mathbf{w} + b) - 1 > 0$, the loss is 0, otherwise the loss is proportional to its distance with respect to the decision boundary.

Combination methods

Two approaches are commonly used in combining the binary classifiers to perform multiclass prediction. The *o*ne-vs-all approach builds k (the number of classes) binary classifiers which distinguish one class from all the other classes lumped together. For a test sample $\mathbf{x}$ the binary classifier outputs form a k-vector $f(\mathbf{x}) = (\mathrm{f_1}(\mathbf{x}), ..., \mathrm{f_k}(\mathbf{x}))$. If $f_i(\mathbf{x})$ is a real number (i.e. predicted label with confidence value), then the predictor finds the maximum of $f_i(\mathbf{x})$ and assigns the sample to the corresponding class label: $\mathcal{F}(\mathbf{x}) = \arg\max_i f_i(\mathbf{x})$. Similarly, the *a*ll-pairs approach builds $\frac{k(k-1)}{2}$ binary classifiers which distinguish a pair of classes: $f(\mathbf{x}) = (\mathrm{f_{1,2}}(\mathbf{x}), ..., \mathrm{f_{k-1,k}}(\mathbf{x}))$. For each class there are k relevant binary classifiers which distinguish it from the other classes. The confidence values of those k binary classifiers are summed up, and the class with the greatest overall confidence is the winning class: $\mathcal{F}(\mathbf{x}) = \arg\max_i \sum_j f_{i,j}(\mathbf{x})$.

Both the one-vs-all and the all-pairs schemes are special cases of a general output coding scheme (Dietterich & Bakiri (1995), Allwein *et al.* (2000)). A code matrix $\mathbf{M} \in \{-1, 0, 1\}^{\mathrm{k} \times \mathrm{l}}$ is a k by l binary matrix, where k is the number of classes and l is the number of binary classifiers used to construct $\mathcal{F}$. Each column in $\mathbf{M}$ defines a binary partition on class labels. It puts some classes at one side of the boundary (+1), some classes at the other side of the boundary (-1), and ignores the remaining classes (0). A binary classifier f_t is built on a partition of the training examples. Each row in $\mathbf{M}$ is called a codeword of a class. It is the binary classifiers outputs of an "ideal" sample from a given class. The binary classifier outputs of an unknown sample are compared to each codeword in $\mathbf{M}$. The class whose codeword has the minimum distance with respect to binary classifier outputs is assigned to the sample. In formal notations,

$$\mathcal{F}(\mathrm{x}) = \arg\min_{\mathrm{r}} \mathrm{d}(\mathbf{M}(\mathrm{r}), \mathbf{f}(\mathbf{x})) \tag{1}$$

$d(\mathbf{M}(\mathrm{r}), \mathbf{f}(\mathrm{x}))$ is the distance between a codeword $\mathbf{M}(\mathrm{r})$ and the binary classifier outputs $\mathbf{f}(\mathbf{x})$. If the binary classifiers output only predicted labels, then the Hamming distance is used:

$$d_H(\mathbf{M}(\mathrm{r}), \mathbf{f}(\mathbf{x})) = \sum_{s=1}^{l} \frac{1}{2}(1 - \mathbf{M}(\mathrm{r}, \mathrm{s})\mathrm{f_s}(\mathrm{x})) \tag{2}$$

The Hamming distance simply counts the number of bits in $\mathbf{M}(\mathrm{r})$ and $\mathbf{f}(\mathbf{x})$ which disagree with each other. Since the the code matrix entries can have three values ($\{-1, 0, +1\}$) but the predictor outputs can only have two values ($\{-1, +1\}$), the definition is slightly different from the conventional Hamming distance. When the codeword has value +1 or -1 on a given bit, the Hammin distance is increased by one if it is different from the binary predictor output. This is the case when the corresponding class is involved with the binary classifier. When the codeword has value 0 on this bit, the Hamming distance is increased by one half no matter what the binary classifier output is. This is the case when the corresponding class is not involved with the binary classifier on this bit (for instance, class 3 is not classified by the pairwise predictor $f_{1,2}(\mathbf{x})$).

If the binary classifier outputs a real number, then the distance metric should take the confidence of the outputs into account. Allwein and Schapire introduced the loss-based distance between a codeword and real-number classifier outputs (Allwein *et al.* (2000)). It is the sum of losses over the outputs of binary classifiers:

$$d_L(\mathbf{M}(\mathrm{r}), \mathbf{f}(\mathbf{x})) = \sum_{s=1}^{l} L(\mathbf{M}(\mathrm{r}, \mathrm{s})\mathrm{f_s}(\mathrm{x})). \tag{3}$$

The loss function is designed to penalize misclassification and takes the (continuous) classifier output into account.

For the weighted voting and kNN algorithms we define the loss as one minus the confidence value, whereas for SVM we apply the hinge loss function.

Hierarchical partitioning algorithm

Constructing the code matrix is the main issue for the output coding scheme. In principle, a good code matrix should satisfy two criteria: the Hamming distances between row vectors should be large enough so that errors can be corrected, and the error rate made by each binary classifier is low. There are several previous works on designing the code matrix, for example, linear error-correcting (Dietterich & Bakiri (1995)), random codes (Dietterich & Bakiri (1991)), and continuous-valued code matrix (Crammer & Singer (2000)). In this paper, we develop an algorithm which incorporates the partitions of low error rates and constructs a decision-tree-like classifier. We call this algorithm the hierarchical partitioning.

The training procedure starts by searching the binary partitions on class labels that yield low (cross-validation) error rates. Instead of exhaustively searching for all 2^k partitions, we only look at partitions which assign one, two or three classes on one side and the remaining classes on the other side.

If the code matrix contains only the partitions of low error rates it may not be able to decipher all classes uniquely: multiple classes may have identical row vector. Therefore, we need to build another code matrix to distinguish the classes which cannot be uniquely decoded at the current level. The procedures of searching valid partitions and building the code matrix continue recursively until all class labels are uniquely deciphered. The classifier built by this method is a hierarchy of multiclass classifiers.

The classification procedure resembles a decision-tree. Starting from the root the test sample is fed into the multiclass predictor associated with the node. If the predicted label is non-degenerate (i.e. there exists a unique codeword for this class), then the classification procedure is completed. If the predicted label is degenerate (several classes have the same codeword), then the predictor at the next level is applied to the sample. The classification procedure continues until the sample is assigned to a unique label. Figure 2 illustrates how the algorithm works with a simplified 4-class problem. The root contains all class labels and a code matrix which partitions them into subsets. Each child contains the labels in one subset. If the node is not a leaf (i.e., it contains multiple classes), then a code matrix is applied to divide the classes into smaller subsets. In this example the code matrix at the root uniquely distinguishes classes 1 and 2 but lumps classes 3 and 4 together. The code matrix at the second level (a single-column matrix) then separates classes 3 and 4.

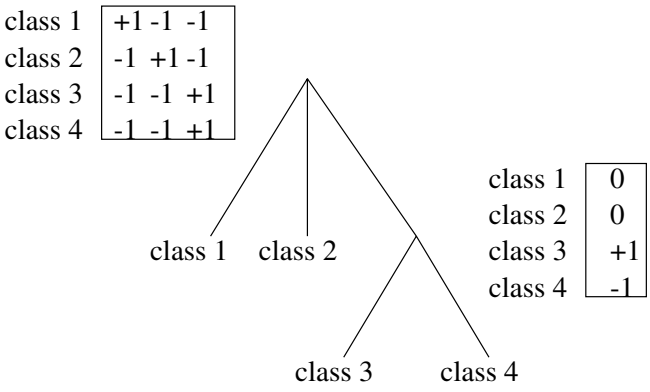

Fig. 2. Hierarchical partitions of a 4-class problem.

RESULTS AND DISCUSSION

Datasets

We collected 190 human tumor samples spanning 14 cancer types. These tumors underwent extensive histopathologic review in order to classify them using traditional methods. This relatively large dataset provides a very useful resource to explore the feasibility of using gene expression to make multiclass distinctions in human cancer. For details about the experimental protocol see the description in the protocols sections of this web site http://www.genome.wi.mit.edu/MPR/. RNA from each sample was hybridized overnight to Affymetrix high-density oligonucleotide microarrays containing probes for 16063 known human genes and expressed sequence tags (ESTs). The arrays were scanned with a Hewlett-Packard scanner, and the expression levels for each gene calculated using Affymetrix GENECHIP analysis software. The data obtained from the arrays was used without re-scaling.

We divided the samples into two datasets: the training set contained 144 samples and the test set contained 46 samples. The compositions of samples from each class are similar in both datasets. Table 1 lists tumor classes and number of samples from each class.

Classification results

We applied three binary classifiers (weighted voting, k-nearest neighbors, support vector machines) in conjunction with two combination scenarios (one-vs-all, all-pairs) on the tumor dataset. The hierarchical partitioning method with the k-nearest neighbors algorithm was also applied. For the weighted voting and kNN features were selected according to the signal-to-noise ratio introduced earlier. Each binary classifier was built on a fixed number of features (genes). The classification results of using 20, 40, 50, 100 and 200 features were reported. SVM chose all genes in the dataset thus feature selection was not applied.

Table 1. Tumor classes.

Index	Tumor type	Abbr.	# training	# test
0	Breast	B	8	3
1	Prostate	P	8	2
2	Lung	L	8	3
3	Colorectal	CR	8	5
4	Lymphoma	Ly	16	6
5	Bladder	BL	8	3
6	Melanoma	M	8	2
7	Uterus	U	8	2
8	Leukemia	Le	24	6
9	Renal	R	8	3
10	Pancreas	PA	8	3
11	Ovary	Ov	8	3
12	Mesothelioma	Ms	8	3
13	Brain	C	16	4

Table 2. Cross-validation errors, 144 samples.

# features	WV		KNN		SVM	
	OA	AP	OA	AP	OA	AP
20	51	45	52	43	-	-
40	45	48	49	51	-	-
50	44	48	48	49	-	-
100	48	54	39	49	-	-
200	48	54	40	47	-	-
All	-	-	-	-	27	61

Table 3. Test errors, 46 samples.

# features	WV		KNN			SVM	
	OA	AP	OA	AP	HP	OA	AP
20	23	20	18	16	17	-	-
40	24	22	22	16	25	-	-
50	26	22	18	17	25	-	-
100	25	25	21	18	25	-	-
200	25	25	17	18	25	-	-
All	-	-	-	-	-	10	21

Table 2 shows the leave-one-out cross validation errors for various methods on the training dataset (144 samples, 14 classes). The one-vs-all SVM outperformed other methods significantly: 27 errors out of 144 samples (18.75%). The one-vs-all kNN with 100 features achieved the second best: 39 errors out of 144 samples (27.08%). There are three interesting observations in comparing the outcomes of various classifiers. First, the average performance of kNN is better than the weighted voting. However, given a fixed feature number kNN does not necessarily outperform the weighted voting. Second, one-vs-all methods tend to achieve lower error rates than all-pairs methods. This is mainly because the binary classifiers in the all-pairs scenario were built from fewer examples than the classifiers in the one-vs-all scenario. In the one-vs-all method, each binary classifier uses all the training samples, whereas in the all-pairs method, the pairwise classifier is trained only on the samples with relevant class labels. The small training set makes the pairwise predictors subject to overfitting. Moreover, irrelevant classifiers in the all-pairs method add extra noise. In addition, for a given sample only k (the number of classes) binary classifiers are relevant to its true label. All the other classifiers generate incorrect answers no matter what the outputs are. Third, there is no clear relation between feature numbers and error rates for the weighted voting and kNN algorithms (at least from the five feature numbers reported here). Unlike SVM, the weighted voting and kNN do not gain by adding more features.

Table 3 shows the prediction outcomes on an independent test set of 46 samples. One-vs-all SVM again outperforms other methods: 10 errors out of 46 samples (21.74%). The second best is all-pairs kNN with 20 or 40 features: 16 errors out of 46 samples (34.78%). In contrast to cross-validation results, kNN is significantly better than the weighted voting. Furthermore, all-pairs methods tend to outperform one-vs-all methods with fixed feature numbers. We do not know what causes the results inconsistent with cross-validation outcomes. The hierarchical partitioning achieves comparable results to the second best predictor for small feature number, but in general it performs worse than the other algorithms. It suggests optimizing the code matrix in terms of cross-validation errors may cause further overfitting to the training set.

Relating confidence to test errors

Tables 2 and 3 show the *hard errors* of classification: an error is counted when misclassification occurs. However, the strength of the prediction is not shown. Loss-based distances between the binary classifier outputs and the codewords represent the strength of prediction. If the loss-based distance with respect to the codeword of the winning class is significantly lower than the other classes, then the strength of prediction is high. On the contrary, if there are multiple or no classes which claim the sample with low loss, then the strength of the prediction outcome is low. Figure 3 illustrates the relation between the confidence and the prediction strength. It shows the binary classifier outputs of 4 test samples using one-vs-all SVM. In the first example only one of the fourteen classifiers returns a positive value, and this value is significantly higher than the other values. We call this a high confidence case since the prediction outcomes are strongly biased toward one class. In the second example only one classifier returns a positive value, but this value is close to 0 and

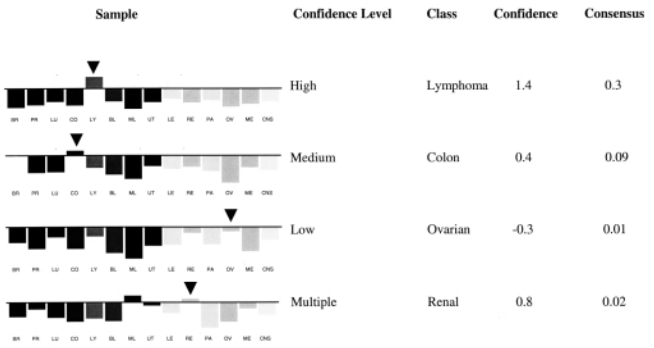

Fig. 3. Binary classifier outputs for four samples.

Table 4. Test errors categorized by confidence values

	OA SVM		OA kNN	
	Error	Total	Error	Total
High conf	0	15	1	15
Med. conf	5	19	3	8
Low conf	5	12	13	23

is not very different from other binary classifier outputs. Although we still assign the sample to the winning class, we are less certain about the classification outcome. We call this a medium confidence case. In the third example, none of the binary classifiers returns a positive value. The winning class is the one which is "less dissimilar" to the test sample. However, it is possible that the test sample falls into a new category (or sub-category) not captured by the training set. In the fourth example, multiple classes claim the sample with positive output values. We call both the third and the fourth examples low confidence cases.

Table 4 shows the test errors for high confidence, medium confidence, and low confidence samples using one-vs-all SVM and one-vs-all kNN classifiers. The results clearly indicate that most of the errors are in the low and medium confidence samples. Among the 15 high-confidence samples in SVM, none of them are misclassified, whereas among the 15 high-confidence samples in kNN, only one of them is misclassified. This strongly suggests that errors tend to appear in the overlapped regions of multiple classes. Samples from those regions are easily confused with other classes.

If we allow the predictor to return "no calls" on low-confidence samples, then the hard error rates are considerably reduced: 5 errors out of 34 samples for SVM and 4 errors out of 23 samples for kNN.

CONCLUSION

In this paper we collected samples from 14 tumor types and generated expression data from those samples. We applied various supervised learning algorithms to classify tumor samples from their expression data. Motivated by the good performance of binary classifiers in tumor classification, we adopted the output coding scheme which combined binary classifiers for multiclass prediction. We applied two commonly used combination strategies: the one-vs-all and the all-pairs methods. In addition, we also developed a hierarchical partitioning algorithm which generated a collection of code matrices and organized them in a hierarchy. We chose three types of binary classifiers as the components of the multiclass predictor: the weighted voting, the k-nearest neighbor, and the support vector machines.

Cross-validation errors and test errors indicate that the one-vs-all SVM achieves the best performance. The kNN outperforms the weighted voting algorithm on average, but in some cases the weighted voting performs better. There is no clear relation between the number of genes in the classifier and the accuracy rate of classification.

By looking at individual classifier outputs, more information is revealed. We categorized samples into high confidence, medium confidence and low confidence types in terms of binary classifier outputs. We found that most of the errors were made by low confidence samples. Very few high confidence samples were misclassified. When we allowed *no calls* to low confidence samples, we were able to reduce the error rate significantly (14.71% for one-vs-all SVM and 17.39% for one-vs-all kNN). This fact suggests most errors occur in the overlapped region rather than the outliers of the class territories.

We suspect the classification accuracy is limited by the number of samples we have. Therefore we are collecting more samples covering broader tumor types. We are also exploring the distinctions between normal tissues and malignant tissues. A comprehensive analysis which covers these aspects will appear soon.

ACKNOWLEDGEMENTS

The authors are grateful to the following colleagues at Whitehead Institute Genome Center: Michelle Gaasenbeek, Christine Huard, Christine Ladd, Shawn O'Malley, and Jane Staunton. The authors are also grateful to the following people who contributed tissue samples and pathology expertise: Massimo Loda, David Waltregny, Phllip Febbo, Margaret Shipp, Raphael Bueno, Kevin Loughlin, Scott Pomeroy, William Gerald, Massachusetts General Hospital Tumor Bank, and the Cooperative Human Tissue Network. The authors thank Professor Tommi Jaakkola at MIT EECS Department for reviewing the draft.

REFERENCES

Alizadeh, A., Eisen, M., Davis, R., Ma, C., Lossos, I., Rosenwald, A., Boldrick, J., Sabet, H., Tran, T., Yu, X., Powell, J., Yang, L., Marti, G., Moore, T., Hudson, J., Lu, L., Lewis, D., Tibshirani, R., Sherlock, G., Chan, W., aand D.D. Weisenburger, T. G., Armitage, J., Warnke, R., Levy, R., Wilson, W., Grever, M. R., Bvrd, J., Bostein, D., Brown, P. & Staudt, L. (2000). Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. *Nature*, **403**, 503–511.

Allwein, E., Shapire, R. & Singer, Y. (2000). Reducing multiclass to binary: a unifying approach for margin classifiers. In *Proceeding of international conference on machine learning*.

Ben-Dor, A., Bruhn, L., Friedman, N., Nachman, I., Schummer, M. & Yakhini, Z. (2000). Tissue classification with gene expression profiles. In *Proceeding of the fourth annual international conference on computational molecular biology*. pp. 54–64.

Breiman, L. (1984). Classification and regression trees. Wadsworth international group.

Crammer, K. & Singer, Y. (2000). On the learnability and design of output codes for multiclass problems. In *Proceeding of ACM conference on computational learning theory*.

Dietterich, T. & Bakiri, G. (1991). Error-correcting output codes: a general method for improving multiclass inductive programs. In *Proceeding of the eighth national conference on artificial intelligence*. pp. 572–577.

Dietterich, T. & Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research*, **2**, 263–286.

Duda, R. (2001). Pattern classification. Wiley.

Golub, T., Slonim, D., Tamayo, P., Huard, C., Gassenbeek, M., Mesirov, J. P., Coller, H., Loh, M. L., Downing, J., Caligiuri, M., Bloomfield, C. & Lander, E. (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, **286**, 531–537.

Hanahan, D. & Weinberg, R. (2000). The hallmark of cancer. *Cell*, **100**, 57–71.

Minsky, M. (1969). Perceptrons: an introduction to computational geometry. MIT Press.

Slonim, D., Tamayo, P., Mesirov, J., Golub, T. & Lander, E. (2000). Class prediction and discovery using gene expression data. In *Proceeding of the fourth annual international conference on computational molecular biology*. pp. 263–272.

Vapnik, V. (1998). Statistical learning theory. Wiley.