Using Classification Trees, Bagging and Boosting, and Random Forest to Detect Cancerous Tissue in DNA Microarray Data

Karl Winterling, Nicholas Sullivan, & Anthony Gusman
Department of Mathematics
California State University
Fullerton, CA 92834, USA
Email: kwinterling@csu.fullerton.edu, nbsullivan@csu.fullerton.edu, agusman@csu.fullerton.edu

***Abstract*—DNA microarray data represents the gene expression levels in a tissue sample as a sequence of numbers. Historically, unsupervised *cluster analysis* methods have been used to investigate DNA microarray data. While these methods are useful for identifying similarly expressed gene groups, they are unreliable in detecting cancer. The accurate classification of potentially cancerous tissue samples is an important biomedical problem with far-reaching implications. Impacts include improved diagnosis ability and earlier detection, which could lead to saving lives. In this paper we investigate the use of decision tree ensemble methods, including bagging, random forest, and boosting, to classify DNA microarray observations (e.g., as either cancerous or non-cancerous).**

We explore five microarray data sets. Any missing values had been imputed prior to receiving the data. The first four consist of cancerous and non-cancerous tissue samples taken from bladder (126 samples, 6688 genes), liver (207 samples, 5520 genes), colon (62 samples, 2000 genes), and prostate (139 samples, reduced to 243 genes). The last is a leukemia data set (73 samples, 7129 genes) consisting of two cancer forms, acute lymphocytic leukemia (ALL) and acute myelogenous leukemia (AML).

Single decision trees are weak classifiers that tend to overfit. Ensemble methods such as bagging, random forest, and boosting intend to transform weak classifiers into strong classifiers. This is achieved through the use of aggregation and/or decorrelation, which also lower the variance in prediction error. Random forest, in particular, is known to be a powerful prediction method. We implement these methods using the popular scikit-learn library for Python.

Error analysis is carried out using k-fold cross-validation to approximate the mean accuracy and generalization error of each method. We carry out several case studies to improve accuracy performance. For random forest, sampling without replacement rather than bootstrapping, and selecting $\sqrt{n}$ features both improve accuracy. For random forest and boosting, we find that stratified sampling improves accuracy. The performance of random forest and boosting begins to peak as early as 20 trees. We also explore a generalization of random forest, namely, extremely randomized trees. Finally, comparing the accuracy of each method, we find that extremely randomized trees performs best on four of the five data sets.

I. Introduction

Correct cancer classification is a significant biomedical problem. Failure to diagnose cancer in a patient may lead to delayed treatment, which may significantly reduce a patient's chances of surviving the disease. On the other hand, a false diagnosis may lead to wasted time and money on treatment and cause a patient unnecessary significant emotional distress and psychological suffering.

Historically, unsupervised *cluster analysis* methods have been used to investigate DNA microarray data. While these methods are useful for identifying similarly expressed gene groups, they are unreliable in detecting cancer [1]. Supervised learning methods, such as classification (decision) trees, are better suited for this task.

II. Methods

Classification trees provide a useful framework for multidimensional supervised learning and data classification. Under certain circumstances, however, raw classification trees may be susceptible to problems such as overfitting that reduce their utility in identifying patterns in complex data. More precisely, by the term "overfitting" we mean that the model does a poor job classifying data outside of its training set.

In a classification problem, a learning set is of the form $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ where $\mathbf{x}_i \in \mathbb{X}$ and $y_i \in \mathbb{Y}$. For our purposes, we focus on binary classification, so $\mathbb{Y} = \{-1, 1\}$.

A. Model Error in Learning Methods

See the relevant sections (especially 7.3) of [2] for more details on what follows. The bias-variance decomposition is perhaps easiest to explain in the context of regression models. Let X be an input to the function f and $f(X)$ the output. We let $Y = f(X) + \varepsilon$ such that $E[\varepsilon] = 0$ and $\mathrm{Var}(\varepsilon) = {\sigma_\varepsilon}^2$. Then if $\hat{f}(X)$ is a regression

prediction of f then the error in prediction given $X = x_0$ may be estimated as,

$$\begin{aligned}\mathrm{Err}(x_0) &= \left(E\left[\hat{f}(x_0)\right] - f(x_0)\right)^2 \\ &\quad + E\left[\left(\hat{f}(x_0) - E\left[\hat{f}(x_0)\right]\right)^2\right] + {\sigma_\varepsilon}^2 \\ &= \left[\mathrm{Bias}\left(\hat{f}(x_0)\right)\right]^2 + \mathrm{Var}\left(\hat{f}(x_0)\right) + {\sigma_\varepsilon}^2.\end{aligned} \tag{1}$$

The bias may be interpreted as the predictor's systematic deviation from the mean while the variance measures the tendency for predicted data to disperse around the mean. We call the term ${\sigma_\varepsilon}^2$ the noise, which is the component of the error that we cannot reduce by making changes in the model. In the case of using bagging or random forest for regression, we take the arithmetic mean of the classifiers, so,

$$\begin{aligned}\mathrm{Var}\left(\hat{f}(x_0)\right) &= \mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n}\hat{f}_i(x_0)\right) \\ &= \frac{1}{n^2}\,\mathrm{Var}\left(\sum_{i=1}^{n}\hat{f}_i(x_0)\right).\end{aligned} \tag{2}$$

Equation (2) suggests that bagging reduces variance in regression modeling. Further benefits are garnered by decorrelating. As in [3], suppose that $\mathrm{Var}\left(\hat{f}_i(x_0)\right) = \sigma^2$ for each classifier and $\mathrm{Cov}\left(\hat{f}_i(x_0), \hat{f}_j(x_0)\right) = \rho\sigma^2$. Then, we may expand equation (2) as

$$\begin{aligned}\mathrm{Var}\left(\hat{f}(x_0)\right) &= \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\mathrm{Cov}\left(\hat{f}_i(x_0), \hat{f}_j(x_0)\right) \\ &= \rho\sigma^2 + \frac{1-\rho}{n}\sigma^2.\end{aligned} \tag{3}$$

Decreasing ρ minimizes the first term, while increasing n minimizes the second term. In practice, the covariance between classifiers may not be equal. Nevertheless, such analysis suggests general performance trends. Random forests can decorrelate individual classifiers via feature selection, thereby improving performance.

Much of the literature on machine learning has focused on extending the regression error model to classification error. As [4] suggests, the regression model for error may be adapted to binary classification, that is, when the range of f is taken (without loss of generality) to be $\{-1, 1\}$.

B. Classification Trees

Our ensemble methods use classification trees as their base estimator. Suppose we have a learning set S, where each vector $\mathbf{x}_i \in S$ consists of P features: $F_1, F_2, \ldots, F_P$ (e.g., expression levels for P genes). A tree classifier $C(\cdot, S)$ recursively partitions S across the features. Several algorithms exist to carry out this partitioning process [5]; scikit-learn uses a version of the CART algorithm:

1) Starting at the root node, scan across the P features and find the "best split"—the point along one of the features that minimizes the "impurity" of the resulting nodes.
2) Repeat this process for each of the child nodes until a stopping criterion is reached.

For example, imagine our data has two classes: "A" and "B." Let $\{\text{A: } a, \text{B: } b\}$ indicate a counts of "A" and b counts of "B" in a set. Then, a partition into $S_{11} = \{\text{A: } 9, \text{B: } 1\}$ and $S_{12} = \{\text{A: } 0, \text{B: } 7\}$ would be preferred over $S_{21} = \{\text{A: } 4, \text{B: } 4\}$ and $S_{22} = \{\text{A: } 5, \text{B: } 3\}$. The idea is that the former are "less diverse" and "easier" to classify than the latter. After the partitions have been formed, a majority rule can be used to classify test observations. If $x \in S_{11}$, we label x as "A," while if $x \in S_{12}$ we label it "B." We have more confidence in this classification if the impurity of the partitions is low.

We measure the impurity with the *Gini impurity*. Suppose there are J classes and let p_i be the probability of choosing an observation with label i. The Gini impurity of a subset is computed as

$$I_{\text{gini}} = \sum_{i=1}^{J} p_i(1 - p_i) \tag{4}$$

which is the probability of misclassifying an observation. In the two-class case, this reduces to

$$I_{\text{gini}} = 2p_1(1 - p_1) \tag{5}$$

which is maximized for $p_1 = 0.5$ and minimized when $p_1 = 0$ or 1. Thus, we favor partitions which yield a smaller Gini impurity. To prevent the formation of trivial subsets, threshold conditions are invoked such as requiring at least n elements in each partition.

C. Random Forest and Bagging

Bagging is arguably the simplest possible stochastic ensemble learning algorithm. In bagging, N learning sets are created by sampling from the learning set with replacement and N classifiers are trained on the N training sets. Random Forest is an extension of bagging that uses decision trees as classifiers. At each node in training a decision tree, a random forest algorithm samples from the features in the learning set without replacement and uses only those features in determining the "best split" [6].

A major problem with decision trees is that they tend to overfit the learning set. That is, the tree builds a complicated structure that accurately classifies data similar to the learning set but this accuracy does not generalize to newly observed data. Intuitively, random forest and bagging reduce overfitting by producing a large number of simpler structures that classify based on plurality vote. Randomness helps ensure that the final structure is not overfitted to the learning set, so its performance on the learning set should be closer to the generalization error. See [6] for a more detailed

discussion of random forests, including a rigorous proof that they converge.

D. Boosting

Boosting is a method that, like random forest and bagging, constructs a learner using a number of base learners. A boosting algorithm trains N learners on different subsets of the learning set and then weights them based on prediction performance. The weighting determines how much importance is assigned to each trained learner's decision. By way of analogy, suppose a number of mathematicians are given a difficult True/False problem. They cannot talk to each other about the problem but they can discuss an optimal weighted voting strategy for determining their final response to a statistically likely problem based on each mathematician's background (for more motivation, watch [7]).

The purpose of a boosting algorithm is to compute such an optimal voting strategy based solely on each classifier's performance on some part of the data. In this paper, we restrict ourselves to the AdaBoost algorithm. We describe AdaBoost based on the presentation in [8]. AdaBoost works by training N classifiers in sequence so that the tth classifier is trained on data weighted in favor of parts of the data that the $t-1$st classifier failed to predict. For the final result, each classifier is weighted according to its overall accuracy. We start giving each part of the data set equal weight.

Represent the weighting of the data set by D_t at step t. To start, we weight each datum equally, so $D_{1i} = 1/m$ where m is the number of data points. At the tth step, we use D_t to train a classifier C_t on the learning set weighted by D_t. We then compute the weighted error ε_t by summing the weights in D_t corresponding to data points that C_t fails to classify when tested on the unweighted learning set. Then we define α_t by,

$$\alpha_t = \frac{1}{2}\log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) = \log\left(\sqrt{\frac{1}{\varepsilon_t}-1}\right) \tag{6}$$

So α_t is monotonically decreasing as a function of ε_t. Then we define the weights for the next iteration by letting $D_{(t+1)i} = D_{ti}e^{-\alpha_t}$ if $C_t(\mathbf{x}_i) = y_i$ and $D_{(t+1)i} = D_{ti}e^{\alpha_t}$ and then normalizing so that $\sum_{i=1}^{n} D_{(t+1)i} = 1$. After N steps, the final classifier is defined by,

$$C(\mathbf{x}) = \operatorname{sign}\left(\sum_{t=1}^{n} \alpha_t C_t(\mathbf{x})\right) \tag{7}$$

To gain insight into why AdaBoost improves performance over classification trees, we follow chapter 10 of [2]. Define,

$$L(y, C(\mathbf{x})) = \exp(-yC(\mathbf{x})) \tag{8}$$

So L is small if $y = C(\mathbf{x})$ and large if $y \neq C(\mathbf{x})$ is small because the set of labels is $\{1,-1\}$. It can be shown that, under suitable conditions, AdaBoost is equivalent to a recursive process that trains the ensemble by adding the classifier C_t that minimizes L with respect to the ensemble at each step. AdaBoost thus gives the benefit of both a collection of simpler classifiers to reduce overfitting and reduced error as N increases.

Indeed, as discussed in [2], we may define the forward stagewise additive modeling algorithm. Evolve a classifier by starting with $G_0(\mathbf{x}_i) = 0$, which we may interpret as an undecided classification. Let b be a function. Then iterating from $m = 1$ to $m = M$, the Mth step is,

$$\begin{aligned}(\beta_m, \gamma_m) &= \arg\min_{\beta,\gamma} \sum_{i=1}^{n} L(y_i, G_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i, \gamma)) \\ G_m(\mathbf{x}_i) &= G_{m-1}(\mathbf{x}_i) + \beta_m b(\mathbf{x}_i, \gamma_m)\end{aligned} \tag{9}$$

As shown in chapter 10 of [2], AdaBoost is equivalent to (9) where G_m is the final classifier at step m and b is the classifier C with weighting determined by γ_m. If we solve (9), we may show that $\beta_m = \alpha_m$ as in (6) and that the update rule D holds. See section 10.4 of [2] for derivations.

III. Study Design

A. Data Sets

The data sets explored are summarized in Table I. The ratio column describes the proportions of normal to cancerous (ALL to AML in the leukemia set) classes in each data set.

Data set	Genes	Samples	Ratio
Liver	5520	181	76:105
Bladder	6688	125	22:103
Leukemia	7129	73	25:48
Colon	2000	62	22:40
R. Prostate	243	139	52:87

TABLE I: Datasets used in experiments

B. Evaluating Performance

Our primary objective is to explore the predictive power of tree-based ensemble learning methods. To do this required investigation into optimal data interaction for each method, quantification of ensemble performance versus that of a single classifier, and comparison of different ensemble methods. We performed three case studies addressing these questions:

1) What are the best data parameters for each method?
2) How do ensemble methods compare with single classifiers?
3) Which ensemble method performs best?

Due to time limitations, an exhaustive evaluation and documentation of all proposed ensemble learners could

not be included in this report for each case study. Nevertheless, our initial results are promising and reported in Section IV.

To validate our results, we use 10-fold cross-validation. The purpose of cross-validation is to better estimate true performance by preventing overfitting. In the case of a 10-fold cross-validation, this is achieved by randomly assigning the data to ten equally-sized (roughly) subsets, setting aside one of these as a test set, using the remaining nine as a training set, and rotating through all ten possible test sets. Such a process yields a distribution of performance statistics giving better insight into the method's general performance than a simple "holdout" test. The size of the respective training and test sets for each data set are given in Table II.

Dataset	Train	Test
Liver	163	18
Bladder	112	13
Leukemia	66	7
Colon	56	6
R. Prostate	125	14

TABLE II: Size of sets in 10-fold cross-validation

C. Algorithms

All results were computed with the scikit-learn Python package [9]. Code is available at: https://github.com/angusman/canClassTrees

IV. Results

A. Case Study 1 (Data Tactics)

In our study of tree-based methods, we noted two recurring "tactics" in learning data:

- Bootstrapping data (random forest, bagging, and boosting)
- Random feature selection (random forest)

These tactics provide several choices. When sampling data, one can choose to use bootstrapping or sampling without replacement (SWOR). For random feature selection, one can specify the number of features to draw. We decided to explore the effects of these options on random forest's performance since it was the only method that incorporated both tactics.

The performance of random forest (100 trees) using bootstrapping and SWOR is compared in Figure 1. For consistency, the same train or test sets are used for each run. We observe that SWOR yields average accuracy as good or better than bootstrapping. Moreover, our data indicated that the variance also tended to decrease when running SWOR. This could be due to the smaller chance of forming redundant sets when using SWOR versus bootstrapping which tends to form sets with about two-thirds unique samples [6].

Next, we investigated four choices of feature size: $\log_2(n)$, $\frac{1}{2}(\log_2(n)+\sqrt{n})$, $\sqrt{n}$, and $\sqrt{2n}$, where n is the maximum number of features available. These settings specify four different forests (each of 100 trees). As before, the same train or test sets are used for each forest. Figure 2 displays the results. The changes in average accuracy only seem dramatic for two of the five data sets, and the default parameter $\sqrt{n}$ seems to reliably perform as good or better than other choices across data sets.

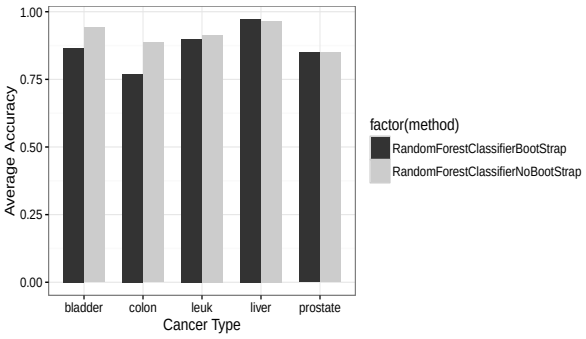

Fig. 1: Avg. accuracy of random forest with and without bootstrapping

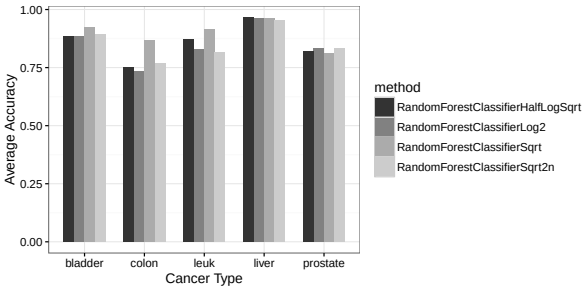

Fig. 2: Avg. accuracy of random forest with varying features

We considered a modification to the cross-validation scheme known as *stratified cross-validation*. The data is assigned to subsets such that the proportion of classes is preserved as much as possible. We conjectured that such a property might help eliminate extreme situations (such as a learning set consisting entirely of one class!) and improve performance. We tested this conjecture using random forest and boosting (bagging was omitted in this case study since it is theoretically similar to random forest). Indeed, we observe some improvement in prediction accuracy for these methods in Figures 3 and 4.

Our findings in this first case study influenced our experiment design for the remaining case studies as shown in Table III.

Sampling:	bootstrapping	(default)
Random forest nfeatures:	$\sqrt{n}$	(default)
Stratified sampling:	True	

TABLE III: Experiment settings for case studies 2 and 3

The choice to use the default settings for bootstrapping even though SWOR showed better accuracy was made

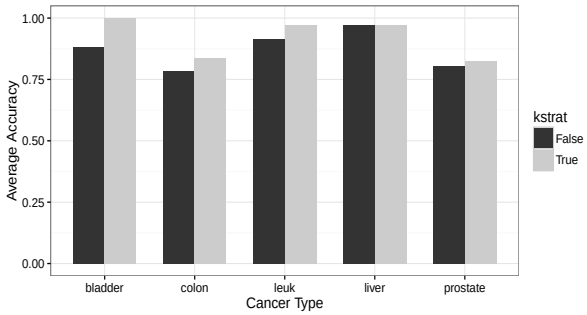

Fig. 3: Avg. accuracy of random forest with and without stratified sampling

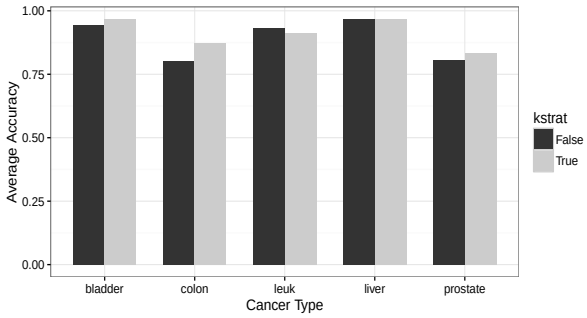

Fig. 4: Avg. accuracy of AdaBoost with and without stratified sampling

for several reasons: 1) SWOR is more computationally expensive than bootstrapping, 2) SWOR was not tested for bagging, 3) stratified sampling yields similar improvements (and was tested for more methods). Further research into the efficacy of SWOR for bagging, and the consequences of using stratified sampling and SWOR together would be interesting.

B. Case Study 2 (Ensemble vs. Solo)

In this case study, we study the effects of using larger ensembles on prediction accuracy and variance. After fitting and predicting with a single decision tree, we compared results using both random forest and boosting with 1, 2, 5, 10, 20, 60, and 100 trees. The same training and test sets were used throughout the experiment. Results are displayed in Figures 5 and 6. We

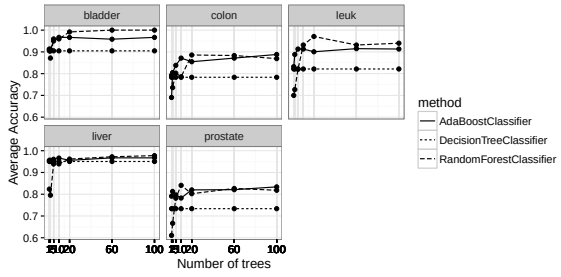

Fig. 5: Avg. accuracy of ensemble methods vs. decision tree

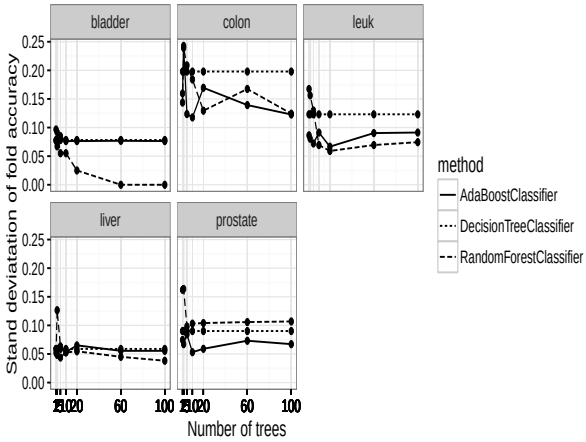

Fig. 6: Standard deviation of ensemble methods vs. decision tree

found that random forest variance decreases as predicted, usually outperforming that of a single decision tree. On the other hand, boosting variance tends to remain constant. Both random forest and boosting outperformed a single decision tree in terms of mean accuracy. Gains in accuracy tended to level off past 20 trees.

One reason boosting may not be reducing variance is suggested by the partial derivatives of Equation 3:

$$\begin{aligned} \frac{\partial \operatorname{Var}\left(\hat{f}(x_0)\right)}{\partial n} &= -\frac{1-\rho}{n^2}\sigma^2 \\ \frac{\partial \operatorname{Var}\left(\hat{f}(x_0)\right)}{\partial \rho} &= \left(1-\frac{1}{n}\right)\sigma^2. \end{aligned} \tag{10}$$

If ρ is positively related to n, then the increase in correlation can theoretically outweigh the benefit of a larger n. With a σ on the order of 1×10^{-1} as in our experiments, the benefit of doubling n is on the order of 1×10^{-3}. On the other hand, for $n > 10$, the effect on variance due to ρ is nearly on the order of 1×10^{-1}. The process used in boosting to sample training sets explicitly depends on the accuracy of the prior classifier, which could easily be introducing a larger degree of correlation. This might be what is offsetting the reduction of variance due to n. On the contrary, random forest might achieve more benefit since each tree is split on small, random feature sets, which would tend to decorrelate the classifiers.

C. Case Study 3 (Best Methods)

We compare results from random forest, bagging, and boosting. The performance of *extremely randomized trees* [10] is also evaluated. This method is essentially a generalization of random forest. Rather than exhaustively search the set of random features for the best split point, extremely randomized trees choose a *random* split point for each feature, and take the best of those. Each tree-based ensemble method is grown with 100 trees over

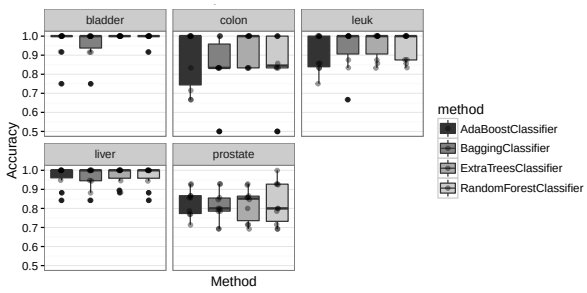

Fig. 7: Boxplots of fold accuracy

the same training and test sets. Table IV summarizes the results for mean accuracy.

Method	Bladder	Colon	Leukemia	Liver	R. Prostate
AdaBoost	96.67%	88.81%	91.31%	96.72%	**83.38%**
Bagging	95.83%	81.67%	93.75%	96.16%	81.16%
Extra Random	**100%**	**93.33%**	**95.65%**	**97.22%**	82.00%
Random Forest	99.17%	83.57%	94.40%	96.69%	82.65%

TABLE IV: Mean classification accuracy. Best results in **bold**.

All methods considered are competitive and exhibit strong performance. Of the ensemble methods, extremely randomized trees performs the best on the bladder, colon, leukemia, and liver data sets, while AdaBoost performs best on the reduced prostate data set. Bagging performs the worst on four of the five data sets. In order to obtain a bigger picture of the each method's performance (which involve stochastic elements), Figure 7 provides a box-and-whiskers plot showing the distribution of classification accuracy for each of the test sets formed by 10-fold stratified cross-validation. We can observe that the methods have differing degrees of variance depending on the data set. The methods appear to struggle the most with the colon and prostate data sets exhibiting lower accuracy and higher variance, perform better on the leukemia data set, and are particularly effective on the liver and bladder data sets. Bagging yields the most consistent results (in terms of variance) across data sets.

V. Discussion

A. Summary

Cancer samples from a variety of data sets are classified using bagging, boosting, and random forest. These methods are implemented with the scikit-learn Python package. Several parameter settings are explored, and performance against that of a single decision tree classifier is quantified. Finally, we introduce the method of extremely randomized trees, and compare the performance of each ensemble method against each other.

B. Contributions

Our work in Case Study 3 highlights the validity of tree-based ensemble methods in biomedical contexts. While these classifiers include an element of chance by their construction, they are very fast ($<$ 2 minutes on a 2.5 GHz MacBook Pro) compared to other prediction methods. This speed provides the advantage of real-time prediction. It also allows them to be run multiple times in succession, generating a distribution of prediction for a test sample, which helps combat their higher variance. We can envision using methods such as random forest in "prescreening tests" for cancer. The results of such tests could then be used to recommend further tests with slower, more deterministic methods.

C. Future Work

Several parameter settings were tested in Case Study 1. Further investigation into bootstrapping without replacement for methods other than random forest is needed. Additionally, we used stratified sampling throughout most of our experiments. It would be interesting to see how sampling without replacement and stratified sampling interact. The performance of the tree classifiers tested in Case Study 3 was promising on the given data sets, however, many more data sets exist and are publicly available. We would like to test the efficacy of the methods on these data sets as well. In this project, our methods were evaluated on the basis of average accuracy and variance. Looking into other performance metrics (e.g., ROC curves) might be insightful. Lastly, we found that extremely randomized trees was a very effective method, often performing as well as or better than random forest. We are curious whether this performance can be optimized further.

References

[1] R. Simon, M. D. Radmacher, K. Dobbin, and L. M. McShane, "Pitfalls in the use of DNA microarray data for diagnostic and prognostic classification," *Journal of the National Cancer Institute*, vol. 95, no. 1, 2003.

[2] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. Springer, 2009.

[3] M. Kalisch, "Random forest," Lecture 10 slides, 2012. [Online]. Available: https://stat.ethz.ch/education/semesters/ss2012/ams/

[4] J. H. Friedman, "On bias, variance, 0/1—loss, and the curse-of-dimensionality," *Data Mining and Knowledge Discovery*, vol. 1, pp. 55–77, 1997.

[5] W.-Y. Loh, "Classification and regression trees," *WIREs Data Mining and Knowledge Discovery*, vol. 1, pp. 14–23, 2011.

[6] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[7] P. Winston, "Learning: Boosting," 2010. [Online]. Available: https://www.youtube.com/watch?v=UHBmv7qCey4

[8] Y. Freund and R. Schapire, "A short introduction to boosting," *Journal of Japanese Society for Artificial Intelligence*, vol. 14, no. 5, pp. 771–780, 1999.

[9] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[10] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, pp. 3–42, 2006.