USING SUPERLEARNER TO PREDICT REMISSION FROM CHILDHOOD EPILEPSY

by

Weifan Yan

Submitted in partial fulfillment of the requirements
for the degree of Bachelor of Statistics, Honours

at

Dalhousie University
Halifax, Nova Scotia
May 2023

Contents

Abstract . . . . . **iii**

Acknowledgements . . . . . **iv**

Chapter 1 Introduction **1**

Chapter 2 Individual Methods **4**

2.1 Logistic Regression 4

2.2 Random Forests 6

2.3 Boosting 9

2.4 Support Vector Machines 11

Chapter 3 SuperLearner **16**

Chapter 4 Discussion **19**

Bibliography **21**

Abstract

This thesis first tries to use several single tree-based classification methods to predict a group of epileptic patients to see if they were still in treatment at the end of the study, and then use the SuperLearner algorithm to combine the single algorithms as an ensemble and to see the performance of the SuperLearner algorithm.

Acknowledgements

I am very grateful for the help I received from my supervisor Professor Smith Bruce, who helped me choose the topic and data for my thesis based on my interests and helped me build the structure of my thesis. Through the weekly meetings, Professor Smith taught me in detail and systematically how to proceed with each step of my dissertation, and patiently helped me to solve the issues when I was confused and when I got stuck in a difficult problem. His guidance gave me my first taste of accomplishment in completing a project and made me want to try academic research in the future, and I appreciate the guidance and help I received from my supervisor. In addition, I am also very grateful to my family and friends for always supporting me and encouraging me.

Chapter 1

Introduction

Approximately half of children with a first unprovoked seizure will have a recurrence. Those with a second seizure are at high risk of further seizures, justifying the diagnosis of epilepsy and daily treatment with antiepileptic drugs. The number of children who become seizure free with treatment is unclear; however, medication can often be successfully withdrawn after several years. For those who have a recurrence after discontinuing medication, the clinical course is largely unknown, and it would be desirable to be able to predict the outcome of epilepsy at the time of diagnosis.

[Camfield et al., 1993] identified a cohort of children with epilepsy in the Nova Scotia population, and followed them to determine long-term outcome. Every pediatric electroencephalographic report from 1977 to 1985 was reviewed, identifying children with a history of one or more possible afebrile seizures. Hospital and pediatric neurology physician charts were then reviewed to limit cases to those with two or more definite afebrile, unprovoked seizures. From 1987 through 1990, follow-up information about the clinical course after diagnosis was obtained. The analysis data set consisted of 504 eligible subjects.

The primary endpoint of the study was whether or not the child was off medication at the end of follow-up, which was taken as a proxy measure for in remission. Stepwise

logistic regression and CART were used to develop predictive models of remission at end of follow up.

A follow up study by [Geelhoed et al., 2005] combined the Nova Scotia cohort with a second prospective cohort of childhood epilepsy patients from the Netherlands. and sought to evaluate the accuracy of a prognostic model based on the two studies combined. Remission was defined as no longer receiving daily medication for any length of time at the end of follow-up. Classification tree models and stepwise logistic regression were used to generate predictive models for the combined dataset, and for the two separate cohorts. The models for each cohort were externally validated on the opposite cohort.

The classification tree model split the data on epilepsy type and age at first seizure. Predictors in the logistic regression model were: number of seizures before treatment, age at first seizure, seizure type, preexisting neurologic signs, and a measure of intelligence. Both classification tree and logistic regression models predicted the outcome correctly in approximately 70% of subjects, with error rate slightly greater in the externally validated cohort.

In this thesis, in order to assess whether prediction accuracy could be improved. several statistical methods have been applied to the Nova Scotia cohort. In order to simplify the analysis, all cases with any missing data were removed, reducing the data set to 422 observations, with 12 predictor variables. The data set was then split into a training data set with 211 observations and a test data set with 211 observations.

In addition to logistic regression, we used boosting, support vector machine, and

random forests to predict the binary outcome off medication at end of followup. We explored the superlearner algorithm, which is an ensemble method that combines combine several machine learning methods, including cross validation. The algorithm generates an optimal weighted average of the included models, and has been proven to be asymptotically as accurate as the best possible prediction algorithm that is included [Van der Laan et al., 2007].

The remainder of this thesis is as follows. In chapter 2 we describe and apply several individual statistical and machine learning methods, including logistic regression, boosting, support vector machines and random forest. In chapter 3 we introduce and apply the superlearner algorithm, and chapter 4 finishes with a summary of results and a few additional comments.

Chapter 2

Individual Methods

This chapter introduces several different methods for predicting a binary outcome, and applies the methods to the epilepsy data set.

2.1 Logistic Regression

Logistic regression is an appropriate method when the data set is moderately lare and the dependent variable is dichotomous. It is similar to multiple regression in that it it dose the same job as other regression analysis do, explains the relationship between dependent variable and and one or more ordinal, nominal, interval or some other types of predictor variables. The types of questions that logistic regression can answer is something like how does the probability of getting lung cancer (yes vs. no) change for every additional pound a person is overweight and for every pack of cigarettes smoked per day? Do body weight or age have an influence on the probability of having a heart attack?

Logistic regression is a reasonable first choice of method to examine the effects of several predictor variables on a binary outcome variable. As with regular regression, logistic regression may have problems when assumptions are not met. Logistic regression has two major assumptions: (1) the outcome varaible should be dichotomous in

nature, and (2) there should be no high correlations among the predictors. The second assumption is quite important and must be carefully asseessed when attempting a causal analysis.

Logistic regression is the simplest model to solve the binary classification problem. The logistic function

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}} \tag{2.1}$$

ensures that the output is between 0 and 1, which is the expected results. After some manipulations, the logistic function leads to

$$log\frac{p(X)}{1 - P(X)} = \beta_0 + \beta_1 X \tag{2.2}$$

where the left hand side is called the log odds, and so the logistic regression model assumes that the log odds is linear in X. To estimate the regression coefficients, we choose the maximum likelihood estimators, i.e β_0 and β_1 which maximize the likelihood function

$$l(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'})) \tag{2.3}$$

Since we have multiple predictors, use the multiple logistic regression model here. Generalizing the log odds for the multiple regression from (2.2) we get

$$log\frac{p(X)}{1 - p(X)} = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p \tag{2.4}$$

We again use the maximum likelihood estimators, which are the parameters maximizing the likelihood. Details of the maximum likelihood estimation, and large sample properties of the resulting estimators may be found, for example, in [Bickel and Doksum, 2015].

Using the monotonicity property of maximum likelihood, the MLE $\hat{p}(X)$ is obtained by substituting the MLE of β in (2.4), and solving. Prediction for an observation X_{test} in the test data set is 1 ("yes") if $\hat{p}(X) \geq .5$, and 0 ("no") otherwise.

Logistic regression was carried out using the "glm" function in R. The following table shows the predictions from the logistic regression when applied to the test data set, together with error rates.

	True	
Predicted	yes	no
yes	27	25
no	49	110
column error rate	0.644737	0.185185

Table 2.1: Test data set performance of logistic regression

The overall error rate is 74/211, or just over 35%. The false negative error rate $\approx$ 65% (proportion of those on medication at study end who are predicted to be off medication) is seen to be much higher than the false positive error rate.

It is important to note that without variable selection, the logistic regression model may include a number of variables which are non-significant. In addition, there may be issues arising from high correlations among the predictors.

2.2 Random Forests

Random forest is a tree-based method that can be used to improve prediction accuracy as compared to using a single classification or regression tree. Random forest produces multiple trees and then combines them to yield a single prediction, and it is improved from bagging.

The purpose of bagging or bootstrap aggregation is to the reduce variance of decision trees, and a normal way to reduce variance is by averaging a set of observations, hence for the statistical method, a normal way to reduce variance is taking many training sets to get different models, using every single model to predict the result, and then average all of the results. However it is not practical to get many different training sets, an alternative way is to bootstrap by taking repeated samples from the single training data set, and generating B different bootstrapped training data sets. For a given observation, record the results for each of the B trees, and the overall single prediction is the result most occurred.

Then for the random forest, the difference from the boosting is the number of choices of predictor when it comes to split in a tree. Random forest decreases the choice of predictor subset size from p, the number of predictor variables to m, which is approximately the square root of p, i.e, $m \approx \sqrt{p}$. And the reason is to decorrelate the trees. Since if there is a strong predictor, and every time when it comes to split we consider the whole predictors, then every time the trees will use the strong predictor to split so that the trees are highly correlated, in which case the variance could not be reduced to the largest extend; however if we only consider m predictors instead of p predictors when it comes to split, then on average the strong predictor will only be used in (p-m)/p times so that to decrease the variance to the largest extend, then it could make the average of resulting trees less variable and hence reliable.

The random forest analysis was carried out using the "randomForest" function in the R library of the same name, with predictions made using the generic "predict"

function.

The following table gives predicton results using the default settings of randomForest.

	True	
Predicted	yes	no
yes	36	29
no	40	106
column error rate	0.526316	0.214815

Table 2.2: Test data set performance of randomForest with default settings.

The randomForest algorithm allows for different settings of tuning parameters, which can be specified using the tuneRF() function in R.

I used tuneRF() the select an optimal value of the mtry parameter, where mtry is the number of predictor variables to be used at each split in the tree.

For the training data set used, with 211 observations, the the default mtry parameter is 3, while tuneRF() selects an optimal parameter value of mtry=4. And the following table shows the test data predictions using the optimal value of mtry, which results in a slightly lower false negative rate than with the default mtry=3.

	True	
Predicted	yes	no
yes	38	29
no	38	106
column error rate	0.5	0.214815

Table 2.3: Test data set performance of tuned randomForest with mtry=4

2.3 Boosting

Boosting is another tree-based method that can be used to improve prediction accuracy from a decision tree.

Boosting works likes bagging, the difference is that bagged trees are independent of each other, while boosted trees are not. Boosting does not use bootstrap sampling to generate trees. Rather, each tree fit by the model uses the information from previous trees, and thus each tree is dependent on prior trees.

The procedure is that given a model, fit a tree using the residual rather than the outcome so that the tree is relatively small with a few terminal nodes. Then the second tree corrects the first tree's error by fitting the first tree's residual. And the third tree corrects the error of the first and second by fitting the residual of the second tree, and so on. By fitting the residual of the preceding tree, we improve the model slowly in the area where it does not perform well, and the predictions are based on the whole ensemble of trees.

There are 3 tuning parameters in the boosting.

The first tuning parameter is the number of trees B, which is the total number of decision trees to create in the ensemble. By increasing the number of trees, we can potentially get better coverage, with the risk of overfitting, and the training time will increase. If set the number of trees to 1, it will just create the initial tree without performing the iteration. The best value of B will be chosen by using cross-validation.

The second tuning parameter is the shrinkage parameter λ, which is a small positive number between 0 and 1 that controls the step size of learning. If λ is too large,

the algorithm might overshoot the optimal solution; however, if the size is too small, it might take much time to reach the best solution. The best value depends on the question addressed and the data set, with typical value used being 0.001 and 0.01.

The third tuning parameter is the number of splits in each tree d, which controls the complexity of the boosted ensemble. By increasing this value, we potentially increase the size of the tree and may get increased accuracy, at the risk of overfitting and increasing training time. Often $d = 1$ works well.

We investigated 2 different settings of shrinkage parameter in this problem.

The following table shows the prediction results with shrinkage parameter set to 0.01

	True	
Predicted	yes	no
yes	34	23
no	42	112
column error rate	0.552632	0.170370

Table 2.4: Test data set performance of boosted trees with shrinkage parameter = .01

and the following table shows the prediction results with shrinkage parameter set to 0.001

	True	
Predicted	yes	no
yes	19	14
no	51	121
column error rate	0.75	0.103704

Table 2.5: Test data set performance of boosted trees with shrinkage parameter = .001

The sensitivity to the tuning parameters is clear, with $\lambda = .001$ leading to considerably fewer test observations predicted as "Yes", with correspondingly very high false negative error rate.

2.4 Support Vector Machines

Support vector machine is an approach for classification that is often considered one of the best "out of the box" classifiers. The support vector machine is an extension of the support vector classifier, and the support vector classifier is improved from the maximal margin classifier. These procedures are described, for example, in [James et al., 2013], chapter 9.

Start from the hyperplane, which is a p-1 dimensional subspace in the p-dimensional space. The equation

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0 \tag{2.5}$$

defines a p-dimensional hyperplane. The hyperplane divides the p-dimensional space into two halves, we can determine on which side a point lies by simply calculating whether the left-hand side of equation (2.5) is positive or negative.

Suppose that we have an n*p data matrix $\boldsymbol{X}$ which consists of n training observations in p-dimensional space, and we want to classify these observations into two classes -1,1, then a proper separating hyperplane is

$$\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip} > 0 \quad if \quad y_i = 1 \tag{2.6}$$

and

$$\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip} < 0 \quad if \quad y_i = -1 \tag{2.7}$$

and we could combine (2.6) and (2.7) in one expression

$$y_i(\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip}) > 0 \tag{2.8}$$

for all i = 1,$\cdots$,n.

We can use the separating hyperplane to construct a natural classifier, that is we classify the test observation x_i based on the sign of $f(x_i) = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip}$. If $f(x_i)$ is positive, then we classify x_i to class 1, and otherwise, we classify x_i to class -1. And if $f(x_i)$ is far away from zero, which means x_i lies far away from hyperplane, we are more confident with the classification; instead, if $f(x_i)$ is close to zero, which means x_i lies closed to the hyperplane, and we are not confident with the classification.

If our data can be perfectly separated by a hyperplane, there exists an infinite number of such hyperplanes because we could shift a little or rotate a little of the hyperplane without contact with any observations. A natural choice of these hyperplanes is the maximal margin hyperplane, which separates the training observations farthest from the hyperplane. The margin is the smallest distance among the perpendicular distances from all training observations to a given hyperplane, and the maximal margin hyperplane is the separating hyperplane in which the margin is the largest. We can then use the maximal margin hyperplane as the maximal margin classifier to classify the test observations. The maximal margin hyperplane is the solution to the optimization problem

$$\max_{\beta_0,\beta_1,\cdots,\beta_p,M}(M) \tag{2.9}$$

$$subject \quad to \quad \sum_{j=1}^{p}(\beta_j^2) = 1 \tag{2.10}$$

$$y_i(\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip}) \geq M \quad \forall i = 1, \cdots, n \tag{2.11}$$

Constraint (2.11) guarantees that each observation will be on the correct side of the hyperplane provided that M is positive. Constraint (2.10) adds the meaning to (2.11), with the constraint of (2.10), the left-hand side of (2.11) gives the perpendicular distance from ith observation to the hyperplane, so that M represents the margin of the hyperplane.

But there are some problems with the maximal margin classifier. First of all, there are cases where there does not exist a perfect separating hyperplane that could separate the classes perfectly, and so we could allow there to be some misclassification errors. Secondly, the maximal margin classifier is extremely sensitive which means that adding an observation could make the separating hyperplane change dramatically and may cause overfitting of the training data. Hence the support vector classifier is improved from the maximal margin classifier. Like the maximal margin classifier, the support vector classifier also classifies a test observation based on which side of a hyperplane it lies. The difference is that the hyperplane is chosen to correctly separate most of the training observations, instead of all training observations, into two classes. The support vector hyperplane is the solution to the optimization problem

$$\max_{\beta_0, \beta_1, \cdots, \beta_p, \epsilon_1, \cdots, \epsilon_n, M}(M) \tag{2.12}$$

$$subject \quad to \quad \sum_{j=1}^{p}(\beta_j^2) = 1 \tag{2.13}$$

$$y_i(\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip}) \geq M(1 - \epsilon_i,) \quad \forall i = 1, \cdots, n \tag{2.14}$$

$$\epsilon_i \geq 0, \qquad \sum_{i=1}^{n}(\epsilon_j) \leq C \tag{2.15}$$

If$\epsilon_i = 0$ then the ith observation is on the correct side of the margin. If $\epsilon_i > 0$ then the ith observation is on the wrong side of the margin. That is, if $\epsilon_i > 1$ then the ith observation is on the wrong side of the hyperplane.

C is a non-negative tuning parameter which is the sum of ϵ_i's, hence it controls the severity of the violation to the margin and hyperplane we could tolerate, and it is chosen by cross-validation.

The support vector machine is an extension of the support vector classifier which enlarges our feature space using kernels in order to accommodate a non-linear boundary between the classes.

$$K(x_i, x_{i'}) \tag{2.16}$$

is some function referred to as kernel that quantifies the similarity of two observations. It could turn out that the solution to the support vector classifier problem involves only the inner products of the observation, where the inner product of two observations $x_i, x_{i'}$ is given by $\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p}(x_{ij}x_{i'j})$

If we take

$$K(x_i, x_{i'}) = \sum_{j=1}^{p}(x_{ij}x_{i'j}) \tag{2.17}$$

this just gives back the support vector classifier and equation (2.17) is known as the linear kernel. The support vector machine is the classifier when the support vector classifier is combined with a non-linear kernel. There are many choices, for example,

we could use a polynomial kernel of degree d

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p}(x_{ij}x_{i'j}))^d \tag{2.18}$$

or radial kernel

$$K(x_i, x_{i'}) = exp(-\gamma * \sum_{j=1}^{p}(x_{ij} - x_{i'j})^2) \tag{2.19}$$

where γ is a positive constant.

The following is the table of test set predictions using default setting of support vector machine as implemented in the R function "svm" of the library "e1071". Note the extreme tendency to predict "no", and the associated very high false negative rate.

	True	
Predicted	yes	no
yes	19	9
no	58	125
column error rate	0.753247	0.067164

Table 2.6: Test data set performance of SVM

Chapter 3

SuperLearner

SuperLearner is a machine learning algorithm which is an ensemble with different kinds of models or the same model with different settings to improve the overall prediction accuracy. In this way, we could then take advantage of the strength of different kinds of models. The theory behind superlearner is presented in [Van der Laan et al., 2007], and the R implementation is described in [Polley et al., 2019].

As with the other methods described in chapter 2, we have begun with our division of the original data set into a training data set and a hold out data set. The SuperLearner algorithm fits training data set using multiple base models. Then, it uses cross-validation to evaluate the performance of each base model on a holdout set of data. Based on these evaluations, SuperLearner assigns weights to each base model, with higher weights given to models that perform better on the holdout data and the overall weight is 1. Finally, SuperLearner combines the base models using these weights to produce a final ensemble model which used to predict new data.

We used the SuperLearner algorithm as an ensemble of 5 basic single model, which are logistic regression, default setting of random forest, random forest with modifed tuning parameter, and 2 different setting of boosting, were the tuning parameters were set at the values used in chapter 2. Unfortunately, the SVM model in the R

superlearner implementation is not functional.

After running superlearner, we wound up with the following ensemble, where Risk is the cross validaated estimate of risk, and coefficient is the weight assigned to the paricular algorithm.

Algorithm	Risk	Coefficient
boosting with shrinkage 0.001	0.2283083	0.4463447
randomForest default setting	0.2341913	0.2991023
randomForest tuning parameter	0.2389952	0.0000000
boosting with shrinkage 0.01	0.2297656	0.0000000
logistic regression	0.2373301	0.2545530

The following table is the test data classification results of SuperLearner.

	True	
Predicted	yes	no
yes	25	18
no	51	117
column error rate	0.671053	0.133333

Table 3.1: Test data set performance of SuperLearner

We can see that the SuperLearner algorithm only choose one setting of model between the same model with different settings, in this case it choose the default random forest setting and the boosting with shrinkage parameter equal to 0.001.

The overall number of misclassification errors, and the false negative and false positive error rates are not dramatically different from several of the single methods discussed in chapter 2.

We used nested cross-validation to estimate the performance the SuperLearner algorithm and to compare it to the single methods.

The following table shows the average risks of all algorithms based on the 10-fold cross-validation, together with standard error, and minimum and maximum estimated risk.

Algorithm	Ave	se	Min	Max
Super Learner	0.2256259	0.011501958	0.1881796	0.2706340
Discrete SL	0.2264714	0.010345281	0.1923071	0.2702207
boosting with shrinkage 0.01	0.2223798	0.009980359	0.1923071	0.2666545
randomForest default setting	0.2286507	0.014948901	0.1782984	0.2951120
randomForest tuning parameter	0.2317188	0.014962817	0.1754516	0.3020568
boosting with shrinkage 0.001	0.2249155	0.009958404	0.1953863	0.2702207
logistic regression	0.2318083	0.013817239	0.2017737	0.2770471

We see that the risks of different individual algorithms are very close, and that the SuperLearner algorithm does not appear to be much better than any of the other single algorithms in terms of risk.

Chapter 4

Discussion

In this thesis we explored several individual methods, and the ensemble method super learner, for predicting a binary outcome variable, with the specific goal of predicting whether or not a patient with childhood epilepsy will be off of medication at the end of followup.

The estimated risks the methods, including super learner, were quite similar, all being within one standard error of the others. In this situation, our choice is to use logistic regression, given the simplicity of the underlying model and the explicit form of the prediction. However, there was no variable selection carried out with the logistic regression, and it is possible that the prediction error rate for that method could be improved by removing unimportant predictors, perhaps by using AIC, a lasso penalty, or some stepwise model building procedure. This is a subject for further research.

One issue concerns the complexity of several of the learning methods. It may be that the size of the training data set (211 cases) is too small to produce reliable predictions without substantial separation between positive and negative cases.

Another issue is the imbalance in the false positive and false negative error rates achieved by the methods. All of the procedures considered have a false negative rate which is much greater than the false positive rate. In this respect, the random forest

method, tuned or not, balances the two error rates better than logistic regression, boosted regression, SVM, or superlearner. A different risk measure might be used in order to balance false negative and false positive rates. This should be done in conjunction with the medical specialist, as there may be very different consequences associated with the two types of error.

In trying to better understand the underlying data, it would be useful to identify that subset of cases which have the same prediction regardless of the method used, and try to determine the associated features/predictors which identify this subset. This is a subject for additional work.

Bibliography

[Bickel and Doksum, 2015] Bickel, P. J. and Doksum, K. A. (2015). *Mathematical statistics: basic ideas and selected topics, volume I.* CRC Press.

[Camfield et al., 1993] Camfield, C., Camfield, P., Gordon, K., Smith, B., and Dooley, J. (1993). Outcome of childhood epilepsy: a population-based study with a simple predictive scoring system for those treated with medication. *The Journal of pediatrics*, 122(6):861–868.

[Geelhoed et al., 2005] Geelhoed, M., Boerrigter, A. O., Camfield, P., Geerts, A. T., Arts, W., Smith, B., and Camfield, C. (2005). The accuracy of outcome prediction models for childhood-onset epilepsy. *Epilepsia*, 46(9):1526–1532.

[James et al., 2013] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.

[Polley et al., 2019] Polley, E., LeDell, E., Kennedy, C., Lendle, S., and van der Laan, M. (2019). Package 'superlearner'. *CRAN*.

[Van der Laan et al., 2007] Van der Laan, M. J., Polley, E. C., and Hubbard, A. E. (2007). Super learner. *Statistical applications in genetics and molecular biology*, 6(1).