Chapter 9: Machine learning

When we learned about microsimulation modeling, we assumed that we were limited to simulating a large population because all we knew about the population were basic summary statistics—the means and standard deviations of key characteristics of people, for example, and the correlation among those characteristics. Increasingly, large datasets of medical research or healthcare data are available for us to sample directly, permitting us to move from simply simulating millions of people to actually sampling from the complex reality of real data on millions of those people. When such data are available, it can be helpful to directly construct a model from the data itself, allowing the subtleties of the data to generate the model. One of the most powerful ways of modeling using large datasets is to use *machine learning.* Machine learning refers to the development of a set of algorithmic approaches—referred to as learners, predictive models, or estimators—that seek to categorize data or predict an outcome. Machine learning encompasses several methods that create models organically from data. The models are created by repeatedly refining rules that define how data are related to an outcome—such as how biomarkers, clinical features, and social circumstances are related to the probability of having a disease. The repeated nature of machine learning is key to the power of machine learning methods; machine learning models are built by sampling over and over again from data, to identify the most accurate way to model a given problem and predict a particular outcome, through repeated refinement. A machine learning approach involves a systematic, iterative process to build a model from the data itself, with the model becoming more accurate by sampling repeatedly from large datasets. Machine learning remains an emerging field; hence, the methods of machine learning that we describe will no doubt become rapidly out-of-date as new methods are devised to perform increasingly-complex tasks from increasingly-expanding datasets. Despite the inherent limitations to writing a Chapter on an emerging field, our purpose here is to provide the basic vocabulary

and foundation for understanding and implementing key machine learning methods for healthcare and public health problem-solving.

Standard logistic regression

Suppose we have access to the data from a large randomized clinical trial. The trial tested whether a new anticoagulant drug (a "blood thinner") was more effective than the standard anticoagulant drug (called warfarin, which is the substance in rat poison) at preventing strokes caused by blood clots in the brain. The new drug was, in fact, found to reduce the risk of stroke on average compared to warfarin, but also had a higher risk of causing severe life-threatening bleeding events. The obvious question among physicians reading the trial results was: which patients are most likely to gain the benefits of lower stroke risk, while having the lowest risk of the side-effect of life-threatening bleeding? (That is, which patients should we prescribe this drug to, instead of prescribing warfarin?) And conversely, which patients have a low likelihood of beneficial stroke risk reduction, while having a heightened risk of bleeding? (That is, among which patients should we absolutely avoid this new drug?)

This problem asks us to calculate *heterogeneous treatment effects*—or systematic differences between people in the effects of an intervention, as compared to the average treatment effect in the trial. While the average person in the trial had a lower risk of stroke and a higher risk of bleeding on the new drug as compared to warfarin, we seek to determine if there are subpopulations of patients with systematically higher benefits and lower risks than the average, or vice versa.

For the purposes of this problem, let's download the data from the book's website called "09_Basu_sampledata.csv", which corresponds to this imaginary trial (note this is not real data, only simulated data for teaching purposes). The trial dataset provides us with a set of patient features, such as demographics of the patients in the trial, their various medical conditions and laboratory values, and their outcomes from the trial (whether or not they had a stroke, and whether

or not they had a life-threatening bleeding event). Note that the data are "cleaned" and nicely organized for the purposes of this Chapter, but more commonly will not be so conveniently provided. For performing data cleaning and organizing of real, messy datasets in *R*, we highly recommend use of the 'tidyverse' package, which is explained in an excellent online guide entitled *R for Data Science* (available at: http://r4ds.had.co.nz/).

After downloading the data, we can import it using the 'readr' package, which reads spreadsheet files such as the comma separated values file (.csv file) that is the format for storing this dataset. We use the read_csv command to import the file and call the dataset 'trialdata', after using the setwd directory to look in the Downloads folder or whatever other folder the data is saved in:

```
> library(readr)
> setwd("~/Downloads")
> trialdata = read_csv("09_Basu_sampledata.csv")
> View(trialdata)
```

The View command will open up a new tab in RStudio to allow us to visualize the data. We could alternatively click on the 'Files' tab in RStudio, navigate to the directory with the data, click the dataset name, and finally select 'Import Dataset' to achieve the same result.

The dataset overall has 10,000 rows (one for each participant in the trial) and 27 variables, which correspond to: systolic blood pressure (sbp), diastolic blood pressure (dbp), serum creatinine (creat; a measure of kidney function), total cholesterol (tchol), high-density lipoprotein cholesterol (hdl), body mass index (bmi), aspartate aminotransferase (ast; a measure of liver function), alanine aminotransferase (alt; another measure of liver function), heart rate (hr), history of diabetes or not (hodm; a dichotomous 0/1 variable), hemoglobin (hgb), prothrombin time (pt; a measure of coagulation), partial thromboplastin time (ptt; another measure of coagulation), platelet count (plts), history of falls (falls; a dichotomous 0/1 variable), history of myocardial infarction (homi), history of stroke (hostr), history of congestive heart failure (hochf), aspirin medication use (asa), thiazide medication use (thiazide), calcium channel blocker medication use

(ccb), angiotensin converting enzyme inhibitor medication use (acei), beta-blocker medication use (bblocker), Lasix medication use (lasix), new episode of stroke while in the trial (stroke), new episode of life-threatening bleeding while in the trial (bleed), and which arm of the trial the participant was in (drug; a dichotomous 0/1 variable with 0 for warfarin and 1 for the new drug). It is possible to quickly identify the distribution of each of these variables by typing:

> summary(trialdata)

The resulting summary tables provide us with the minimum, 1st quartile (25th percentile), median, mean, 3rd quartile (75th percentile), and maximum value of each variable. For dichotomous variables, the mean provides the frequency of having the value of 1. In this particular trial dataset, over 30% of participants had a stroke at some period of the study, while just over half had a life-threatening bleeding event. If we want to see how many had each of the two events, we can create a table of the two outcome variables:

> table(trialdata$stroke,trialdata$bleed)

```
     0    1
0 2727 3603
1 1642 2028
```

Here, the "$" symbol indicates that we want to call a particular variable from within the larger data frame, e.g., calling the 'stroke' variable from within the larger 'trialdata' frame. We see that 2,727 people had neither a stroke nor a major bleed, but 3,603 had a major bleed but no stroke (strokes are in the rows, and bleeds in the columns of the table), 1,642 had a stroke but no bleed, and 2,028 had both a stroke and a major bleed.

We wish to find out what properties of these individuals might predict who is likely to have a stroke but not a bleed, and vice versa. If we simply did a standard logistic regression—that is, identifying how much each individual variable in the dataset correlates to the probability of a stroke or the probability of a bleed, we'd see that no one factor in the dataset appears to definitively determine whether or not a person will have a stroke or have a bleeding event.

Specifically, to create a standard logistic regression model for stroke and another for bleeding in RStudio, we can type in the commands:

```
> logisticmodel_stroke = glm(stroke ~ ., family = binomial(), data = trialdata)
> summary(logisticmodel_stroke)
> logisticmodel_bleed = glm(bleed ~ ., data = trialdata)
> summary(logisticmodel_bleed)
```

The first command creates a new model called 'logisticmodel_stroke' by fitting a generalized linear model (glm) in which we regress the dichotomous outcome of 'stroke' against all other variables in the dataset (specified by the comment "~.", which means we should regress (~) against all other variables (.); we could replace the period with specific variable names (e.g., by typing in "sbp+hodm" instead of ".") if we wanted to just regress against a specific set of variables. The family type of generalized linear regression is binomial, meaning that it is a dichotomous outcome rather than a continuous Gaussian outcome, and the data are called 'trialdata'. When we summarize the resulting model, we see that the variables 'sbp' and 'hodm' are most strongly associated with the outcome of stroke, along with the study drug. Similarly, the commands creating the second 'logisticmodel_bleed' illustrate that 'pt and 'falls' are most strongly associated with the outcome of bleeding, along with the study drug. Specifically, the output from the logistic model for stroke is:

```
> summary(logisticmodel_stroke)

Call:
glm(formula = stroke ~ ., family = binomial(), data = trialdata)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.8218  -0.8141  -0.5174   0.9707   2.4136

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.6941794  0.6173022  -4.364 1.27e-05 ***
sbp          0.0106834  0.0008595  12.430  < 2e-16 ***
dbp          0.0002181  0.0029569   0.074   0.9412
creat       -0.1694172  0.2539632  -0.667   0.5047
tchol        0.0007362  0.0013234   0.556   0.5780
```

```
hdl          -0.0001773  0.0051284  -0.035    0.9724
bmi           0.0099599  0.0085374   1.167    0.2434
ast           0.0058410  0.0127877   0.457    0.6478
alt           0.0096286  0.0134121   0.718    0.4728
hr           -0.0019752  0.0025598  -0.772    0.4403
hodm          1.7448111  0.0512741  34.029   < 2e-16 ***
hgb          -0.0067076  0.0259184  -0.259    0.7958
pt           -0.0225727  0.0272313  -0.829    0.4071
ptt           0.0040074  0.0081911   0.489    0.6247
plts         -0.0002069  0.0009960  -0.208    0.8354
falls         0.0187385  0.0518106   0.362    0.7176
homi          0.0223997  0.0494508   0.453    0.6506
hostr        -0.0488519  0.0656806  -0.744    0.4570
hochf        -0.1023388  0.0506683  -2.020    0.0434 *
asa          -0.0256254  0.0499219  -0.513    0.6077
thiazide      0.0522144  0.0492687   1.060    0.2892
ccb          -0.0057668  0.0505615  -0.114    0.9092
acei          0.0948950  0.0497850   1.906    0.0566 .
bblocker     -0.0952559  0.0504076  -1.890    0.0588 .
lasix        -0.0411248  0.0504110  -0.816    0.4146
drug         -0.7527597  0.0474012 -15.881   < 2e-16 ***
bleed         0.0159633  0.0505442   0.316    0.7521
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 13147  on 9999  degrees of freedom
Residual deviance: 11066  on 9973  degrees of freedom
AIC: 11120

Number of Fisher Scoring iterations: 4
```

In the above output, we see the formula, the residual deviation between the predictions of the model and the actual observed outcomes, and estimated coefficients with standard errors, z values, and P values (Pr(>|z|)) for each coefficient, which the asterisks indicating the degree of significance (P<0.001 for 'sbp' and 'hodm'). The null and residual deviance correspond to how well the outcome is predicted by just the intercept and how well it is predicted by the full model, while the AIC stands for Akaike's Information Criteria, which is a goodness of fit measure. For deviance and AIC, lower values indicate better fit when comparing different models; AIC not only considers fit, but also penalizes

models that have more parameters (see Chapter 11 on over-fitting). If we want the coefficients for each variable as odds ratios, we need to exponentiate the coefficients, using the command exp(coef(logisticmodel_stroke), and if we wish to get the 95% confidence intervals around the odds ratios, we can type exp(confint(logisticmodel_stroke)).

The logistic regression model is useful for assessing what factors might be 'significantly' related to the outcome. We see from the coefficients that people higher systolic blood pressure ('sbp') have a higher risk of stroke (as the coefficient on 'sbp' is positive), people with diabetes ('hodm') also have a higher risk of stroke (as the coefficient on 'hodm' is positive), while those with the new drug ('drug') have a lower risk. Similarly, based on the coefficients for the logisticmodel_bleed model, we see that elevated PT ('pt') and a history of falling ('falls') are related to increased risk of major bleeding, while those with the new drug ('drug') have a higher risk.

CART analysis for personalizing medical treatment decisions

From the logistic regression analysis alone, it is not obvious how we might use this model to answer our initial question: which patients are most likely to gain the benefits of lower stroke risk, while having the lowest risk of the side-effect of life-threatening bleeding? And conversely, which patients have a low likelihood of beneficial stroke risk reduction, while having a heightened risk of bleeding? Is there a particular blood pressure above which we should avoid warfarin and use the new drug? Some specific combination of high blood pressure and low PT value? A combination of all four of these variables?

To answer these questions, it may be useful to adopt a preliminary machine learning approach called *classification and regression tree* (CART) analysis. A classification or regression tree is a data-driven way of constructing a decision tree. Rather than gathering the probabilities of an outcome and drawing a tree for all possible outcomes, as we did in earlier chapters, the CART analysis is automated through the following algorithm: first, let's find which variable

explains the most variation in the risk of stroke (or bleeding), then separate the trial population into subgroups with high versus low values of that variable; next, find the next variable that separates out the population into subgroups, and so on...until we fail to explain much more variation in the risk observed in the data. For example, perhaps high versus low systolic blood pressure separates out the population into high stroke versus low stroke risk, then among the group with high stroke risk, perhaps diabetes is important but among the group with low stroke risk, perhaps another variable is more important. Regression tree analysis uses the data itself to define what 'high' blood pressure versus low pressure is, and which variables separate out the population to construct a decision tree—whether it be blood pressure and history of diabetes, or some other complex combination of variables.

Let's examine the potential uses of a regression tree analysis by way of example. Suppose we want to separate out the population into those trial participants who the new drug *helped* as compared to warfarin (preventing a stroke), and those for whom the drug *harmed* as compared to warfarin (causing major bleeding). We can't observe the same trial participant in both the new drug and the warfarin arms of the trial (that's the 'fundamental problem of causal inference'), but we can compare matched pairs of people who are similar in their various characteristics (their blood pressure, cholesterol, etc.)—one in the new drug arm, the other in the warfarin arm—and see whether one person in the pair had a stroke or not, and whether one person in the pair had a major bleed or not. The people who the new drug *helped* would be those who, in a matched pair, had a stroke in the warfarin arm but no stroke in the new drug arm; the people *harmed* by the new drug would be those who, in a matched pair, had a bleed in the new drug arm but no bleed in the warfarin arm.

To create the matched pairs, we have to install a matching package:

```
install.packages("Matching")
library(Matching)
```

We can generate the matched pairs using the following command, which tries to find pairs of people who are similar in the variables that we found

significant (systolic blood pressure, history of diabetes, history of falls, and PT value). The command specifies that the treatment variable (Tr) is 'drug' and the X variables we want to match on are 'sbp', 'hodm', 'falls', and 'pt':

```
pairs =
Match(Tr=trialdata$drug,X=cbind(trialdata$sbp,trialdata$hodm,trialdata$falls,trialdata
$pt))
```

The command creates a matrix entitled 'pairs', with 'index.treated' as the new drug participant in each pair and 'index.control' as the warfarin participant in each pair. We can check that there is good *balance* (similarity) between the matched pairs by seeing how much difference is left in the key covariates, using the following commands

```
> qqstats(trialdata$sbp[pairs$index.treated], trialdata$sbp[pairs$index.control])
$meandiff
[1] 0.001572816

$mediandiff
[1] 0.001366387

$maxdiff
[1] 0.006636736
```

Here, we see a very small difference (mean, median, or maximum) in systolic blood pressure between each of the matched pairs (less than one-hundredth of one millimeter of mercury). We can similarly check. The other three variables we matched on to find the same type of result.

Now that we created have the matched pairs, we can calculate whether or not people within a pair benefited from the new drug versus warfarin, were harmed by the new drug versus warfarin, or had no benefit or harm. Through the following two commands, we can create a variable for whether or not a person in a pair benefited, had no effect, or was harmed (+1, 0, or -1) in which benefit means having a stroke on warfarin but not on the new drug, and similarly whether or not a person in a pair benefited, had no effect, or was harmed (+1, 0, or -1) in which benefit means having a bleed on warfarin but not on the new drug:

```
strokediff = trialdata$stroke[pairs$index.control] -
trialdata$stroke[pairs$index.treated]
bleeddiff = trialdata$bleed[pairs$index.control] -
trialdata$bleed[pairs$index.treated]
```

Finally, we can create a 'score' in which we sum these entities, so that a person can have a score ranging from +2 (benefited from the new drug in terms of both lower stroke and lower bleeding risk) to -2 (harmed by the new drug in terms of both higher stroke and higher bleeding risk):

```
> score =  strokediff + bleeddiff
> table(score)
score
  -2   -1    0    1    2
 174 1292 2215 1242  200
```

We see that there are less than 200 people who were seriously harmed, but 1,292 who had some harm from the new drug; conversely, only about 200 had benefit in terms of both stroke and bleeding but about 1,242 had at least one or the other benefit.

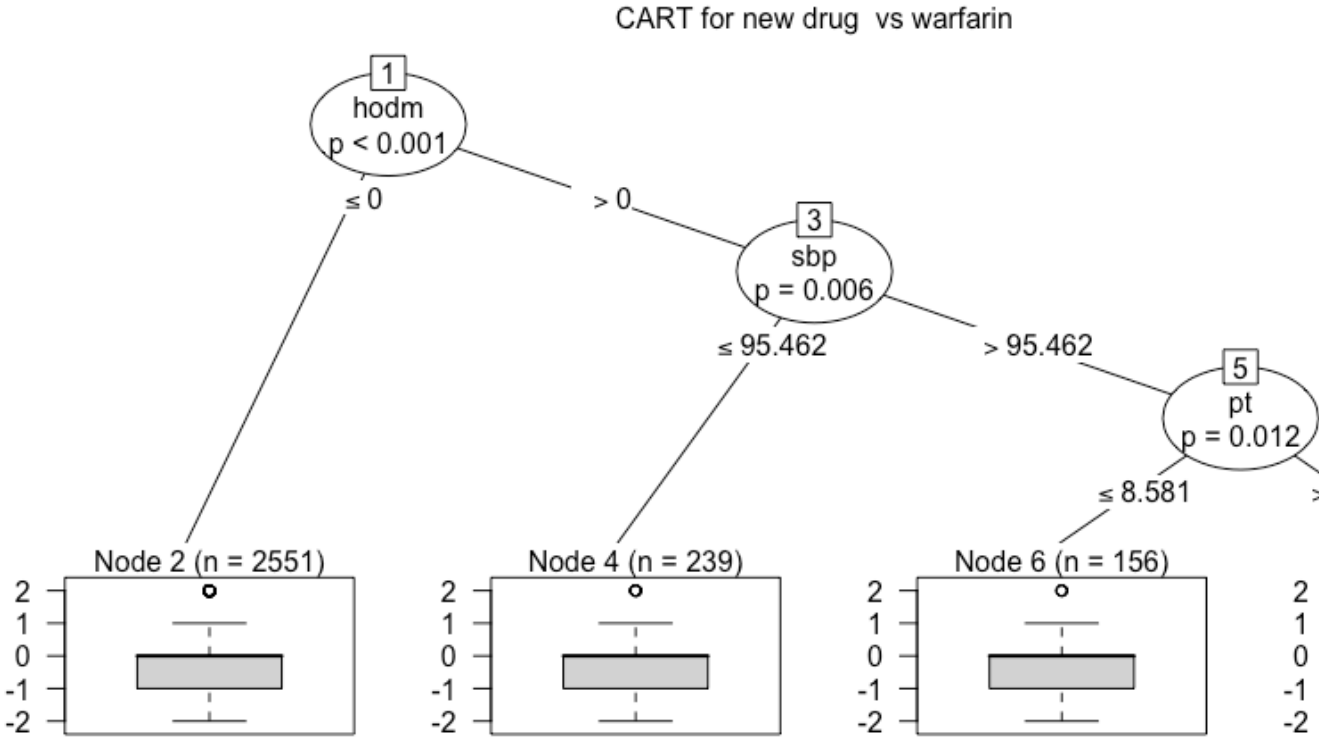
What are the characteristics of the participants who have higher scores (more benefit than harm from the new drug) or lower scores (more harm than benefit from the new drug)? We can perform a CART analysis using the following commands, and create a decision tree that shows us what characteristics critically predict benefit or harm:

```
install.packages('party')
library(party)
matchdata = cbind(score,
                  trialdata$sbp[pairs$index.control],
                  trialdata$hodm[pairs$index.control],
                  trialdata$falls[pairs$index.control],
                  trialdata$pt[pairs$index.control])
colnames(matchdata) = c("score","sbp","hodm","falls","pt")
fit <- ctree(score ~ sbp+hodm+falls+pt, data=as.data.frame(matchdata))
plot(fit, main="CART for new drug  vs warfarin")
```

In the above code, we install the 'party' package (where the 'part' comes from the word "partitioning" or separating the data), then create a new dataset

(binding together columns with cbind) containing the scores, 'sbp', 'hodm', 'falls', and 'pt' values from the control arms (which are highly similar to those in the treated arms given the matching). We then label the new dataset with colnames, and perform CART using the ctree command, which tries to partition the variation in the score based on the variables we provide it (sbp+hodm+falls+pt). Note the data must be put as a "data frame", which is accomplished by the command as.data.frame.

The plot command shows the resulting decision tree, plotted in Figure 9.1. We see that the tree separates out people who do not have diabetes, who had the lowest score. Among those with diabetes, the subset with low systolic blood pressure had the next highest scores. Among those with diabetes and higher systolic blood pressures, those with low PT values had the next higher scores. Finally, the highest scores were among those with diabetes, high blood pressure, and high PT values.



In other words, we may want to reserve the drug for those patients with diabetes, high blood pressures, and high PT values, while using warfarin in other patients.

Looking at the boxplots, though, the result is not very satisfying. We'd like to see one end of the tree with very low values and one end with very high values—that is, see small within-group variation in the score and large between-group variation in the score. But in our decision tree, even those on the far right (highest score) group still had many participants with score values less than zero.

With CART analysis, as with regression in general, we can only consider a few variables and interactions for a reasonable, easy-to-interpret score to be produced. But what if we could have our computers calculate across a number of variables, with the decision tree being far more complex and nuanced, and potentially explaining greater variation in the outcome?

In the next few sections, we'll move beyond simple CART analysis to a more complex set of machine learning methods that try to consider far more than four variables and far more complex non-linear interactions than we considered here, with the potential of gaining greater insight for predicting which patients may be most likely to benefit or be harmed by a given intervention. Before we embark on those more complex methods, we should first agree on some common terms and approaches to define machine learning methods.

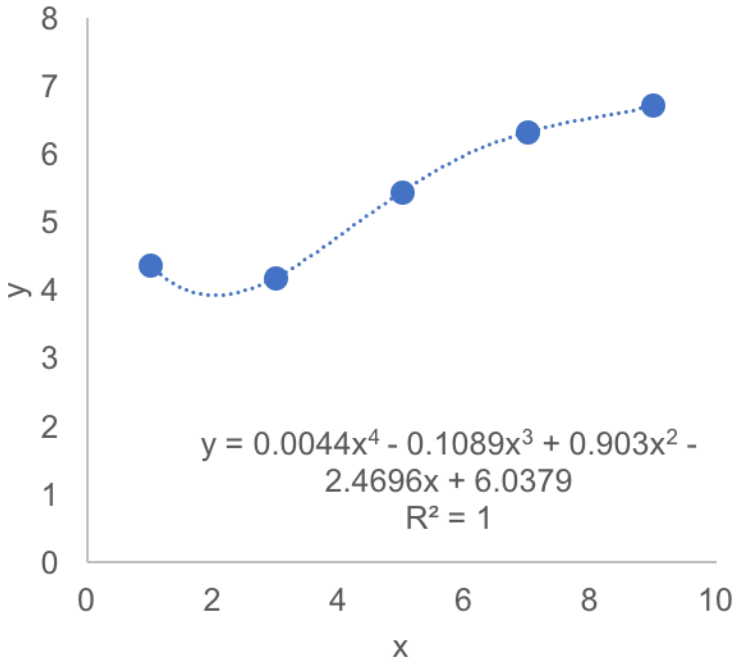
Key terms and concepts in machine learning

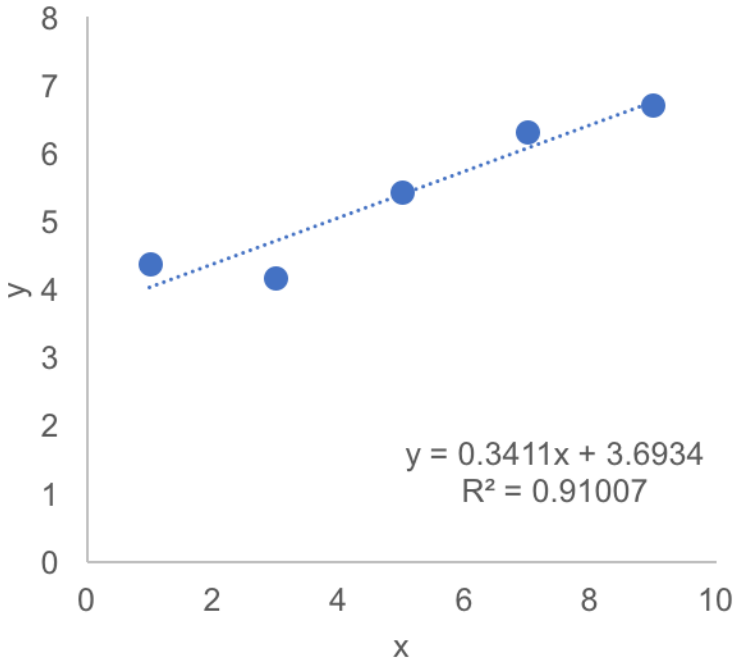
When moving beyond a single CART analysis, machine learning theoreticians generally talk about machine learning algorithms by describing them under various categories: supervised or unsupervised, regularized or not, cross-validated or not, and bagged or boosted.

Supervised versus unsupervised learning. There are two major classes of machine learners. "Supervised" learners are those for which we have a dataset with both input data and outcomes data, such as patient features as inputs and a disease outcome such as stroke or bleeding in the trial dataset we used earlier.

Standard logistic regression and CART analysis are both examples of supervised learning methods, as they translate covariates like blood pressure or history of diabetes into the probability of an outcome. The goal of supervised learners is to learn the relations between input and outcomes data, then predict future outcomes based on future patient's input data. Supervised learners will be the exclusive focus of this Chapter. "Unsupervised" learners, by contrast, are those that learn how to cluster or categorize only input data, providing natural groupings of similar data types, such as through factor analysis and other subjects that are less commonly used in public health or healthcare systems research, and more often in laboratory sciences (such as genetics) or behavioral sciences (such as psychology). In the above example research problem, a supervised learner would be one that uses patient features to predict who will have a stroke or a bleed, while an unsupervised learner would be one that clusters patients into groups based on having similar features (e.g., similar blood pressure values, etc.) analogous to extending our matching algorithm to produce large subgroups instead of just matched pairs.

Overfitting, regularization and cross-validation. "Overfitting" refers to the process by which a learner not only captures the general relationships between input and output data, but wrongly captures random noise in the dataset (Figure 9.2 versus Figure 9.3), which prevents the learner from making accurate and generalizable predictions when applied in the future. In the above example research problem, overfitting might occur if a learner only learns that people who have a systolic blood pressure between 120.1 and 120.9 mmHg with high values on three other biomarkers are at risk for having a stroke, but fails to identify that a person with similarly elevated biomarkers and a systolic blood pressure of 120.0 or 121.0 mmHg are at a similar level of risk.



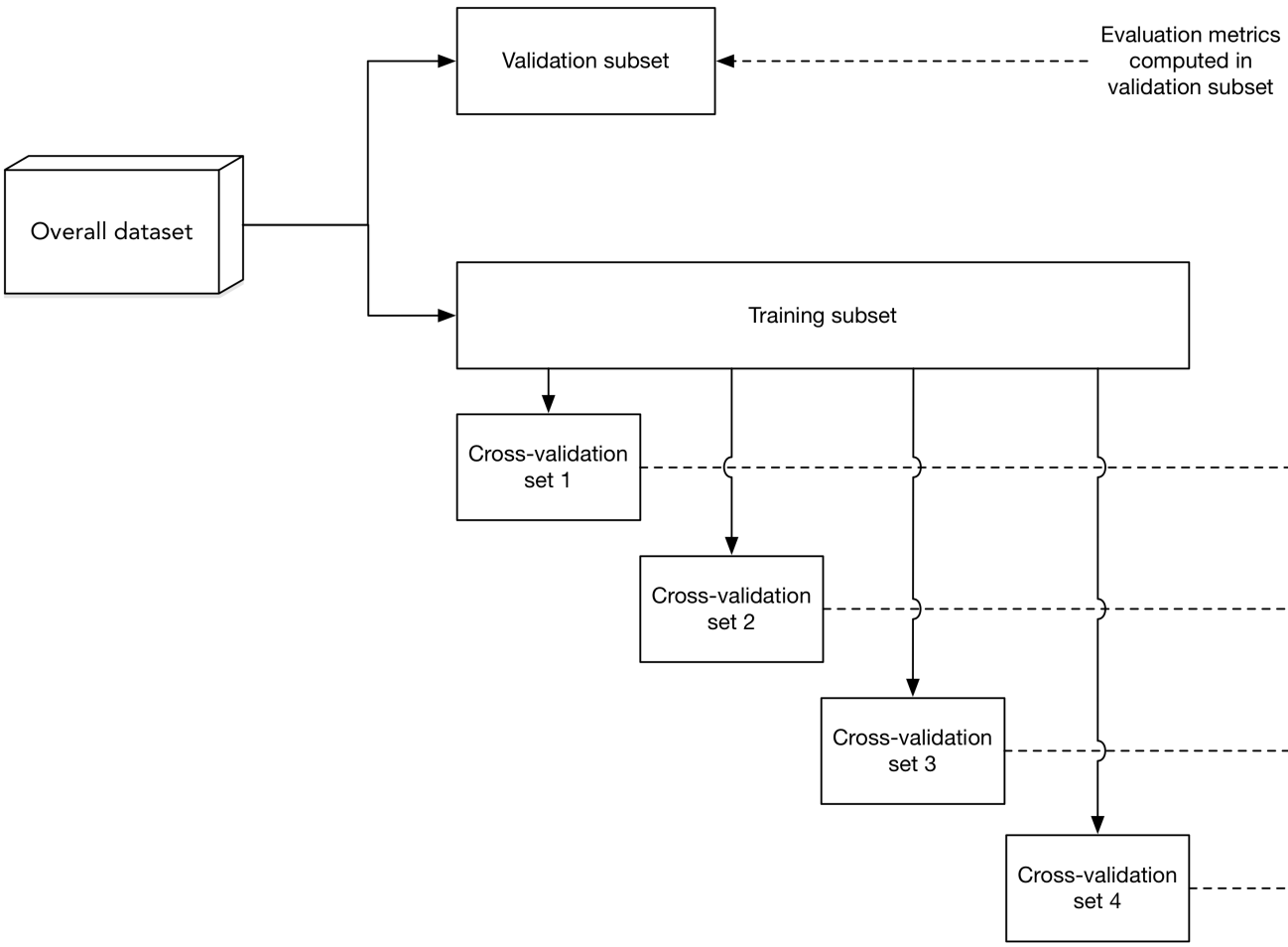


To increase generalizability and prevent overfitting, we try to follow Occam's Razor: that is, we don't want a complicated algorithm to do something a simpler algorithm can do well. A simpler algorithm may be more generalizable,

require less input data or complicated data transformations, be more interpretable, and be more easily fixed if predictions turn out to be error-prone.

"Regularization" is the key process that enforces Occam's Razor when training a machine learner. Regularization is particular important when there are many correlated variables in the input data, as is the case with clinical data (e.g., systolic and diastolic blood pressure are often highly correlated to each other). Two common regularization processes are called: (i) L1 regularization, or "least absolute shrinkage and selection operator" (commonly called "LASSO"), and (ii) L2 regularization, commonly called "ridge" regularization. LASSO selects one among many highly correlated variables to generate a parsimonious model, by penalizing the absolute sum of the regression coefficients so that very correlated variables have coefficients of zero except for one representative covariate. Ridge sets coefficients of correlated variables to be similar values, to minimize the influence of outliers by penalizing the squared sum of the regression coefficients.

In addition to regularization, most people training machine learning algorithms will test for whether they have caused an over-fitting problem by splitting their data into two portions. One portion, called the "validation" dataset, is used only once, after the learner has been fine-tuned, to assess the generalizability of the learner. The validation subset is treated like a reserve wine, opened only once and drunk after all preparations have been made. But we repeatedly sample the other subset, known as the "training" subset, for a process called "cross-validation" (Figure 9.4).



The training subset of the data is our workhorse, our table wine for daily consumption and utility. "Cross-validation" involves repeatedly sampling from a training subset of the data to ensure that our learner is generalizable across multiple subsamples, and to choose how much we want to regularize the learner by selecting a penalty parameter that determines how much to choose one correlated variable over others (via LASSO) or restrict correlated variables' coefficients to be similar (via ridge). After each sample of training data is obtained, we train the learner by finding the penalty parameter value that minimizes the mean-squared error between the predictions from the learner and the observed training data subset.

Many people performing regularization will select a combination of both LASSO and ridge regularization, finding what combination of the two processes

minimizes the error overall across all cross-validations. Combining a bit of both LASSO and ridge is called elastic net regularization. Elastic net regularization seeks to minimize a penalty function over a grid of values for the regularization parameter λ, which controls the strength of the balancing parameter α between LASSO and ridge regression (the second part of the expression below). Specifically, for a logistic regression, the approach seeks to minimize the negative binomial log-likelihood expressed in Equation 9.1:

[Equation 9.1]

$$\min_{(\beta_0,\beta)\in\mathbb{R}^{p+1}} -\left[\frac{1}{N}\sum_{i=1}^{N} y_i \bullet (\beta_0 + x_i^T\beta) - log(1 + e^{(\beta_0 + x_i^T\beta)}\right] + \lambda\left[\frac{(1-\alpha)||\beta||_2^2}{2} + \alpha||\beta||_1\right]$$

The *R* code accompanying this Chapter demonstrates how to find the optimal λ value per the above equation, at each α value, across internal cross-validations on a training subset of data.

Bagging and boosting. A central principal of machine learning is to improve our predictions by making multiple repeated comparisons between our learner's predictions and subsamples of our training dataset. There are two common strategies to maximize our learner's predictive performance. One strategy, known as "bagging", involves training lots of learners on lots of subsamples of the training data, assuming that doing the process over and over again will help weed out extreme outlier predictions and help us settle to a good average prediction (like filling a big bag full of learners and taking the average of them). If there are many possible ways to find an optimal decision tree, for example, bagging can help us avoid getting stuck with one type of tree and help find others that better explain the variance in the outcome.

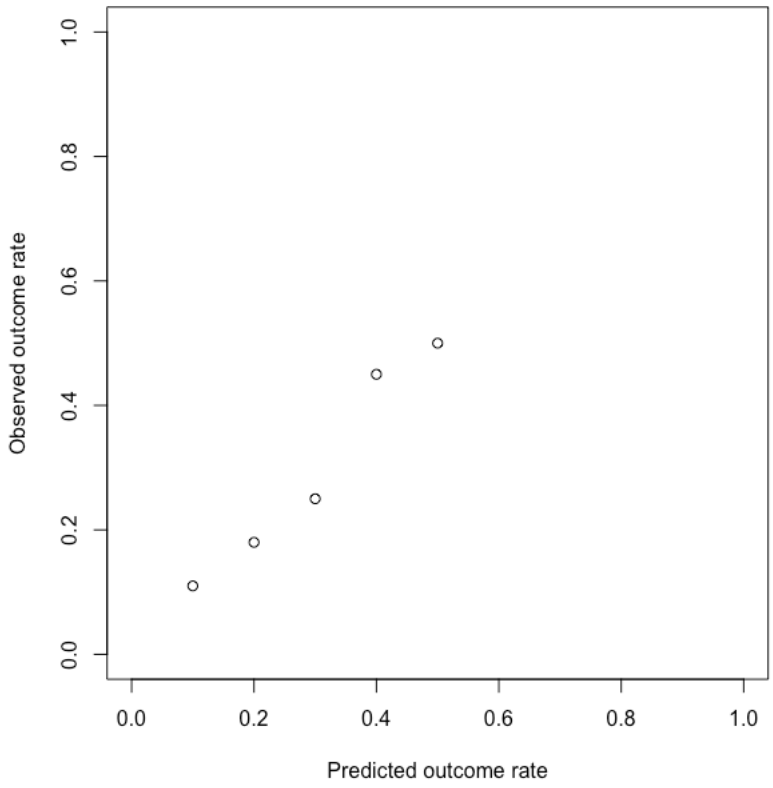
A common alternative strategy called "boosting" involves training one learner, then examining the learners' errors to make a more optimal learner in the next round of sampling, and so on. Simpler problems without a lot of

covariates, and with simple dichotomous or continuous outcome variables, are often well-modeled by boosting.

Rigorously evaluating machine learners

The most common machine learning evaluation metric is the C-statistic, also known as the "area under the receiver operating characteristic (ROC) curve", which we computed in an earlier Chapter when discussing the evaluation of sensitivity and specificity. Just as with a new screening test, a machine learner will have a particular sensitivity and specificity for predicting the outcome. As we discussed earlier, a C-statistic tells us whether the learner correctly selects the higher-risk person among a pair of people (a metric of "discrimination" between higher- and lower-risk people).

In many cases, we don't simply want to distinguish between higher-risk and lower-risk people, but actually want a correct estimate for the absolute risk a person might have for an outcome. In predicting the risk of bleeding from our trial dataset, for example, we want to know whether a 'high' risk of bleeding is only 1% (because a 'low' risk is extremely low) or whether a 'high' risk of bleeding is 50% (which may mean no one wants to encounter that risk). To assess how well a learner assesses real risk, we can create a calibration curve (Figure 9.5), which plots the predicted rate of outcomes among centiles of risk from the validation dataset population (x-axis) against the observed rate of outcomes among those centiles of the validation dataset population (y-axis). A perfect learner will have a 45-degree line between the predicted and observed outcome rates. The Hosmer-Lemeshow test is a common statistical test for evaluating the degree of error between the predicted and observed rates of the outcomes plotted in the calibration curve.



Other often-used metrics in the machine learning literature include the "confusion matrix", which epidemiologists commonly call a contingency table—a 2-by-2 table of true and false positive and negative outcomes in the validation dataset (from which we can calculate sensitivity, specificity, and positive/negative predictive values). Some machine learning papers also use the metric of "accuracy", which is the sum of true positives plus true negatives, divided by the total number of predictions. Others use the "F1 score", which is a weighted average of the positive predictive value (also called "precision") and sensitivity (also called "recall"). In all cases, the pre-specified metric that is chosen for evaluation should correspond to the ultimate application that the learner will be used for, by selecting a metric that corresponds most to the goals of end-users. We show how to quickly and easily calculate the most common metrics, in the examples below.

Major machine learning families

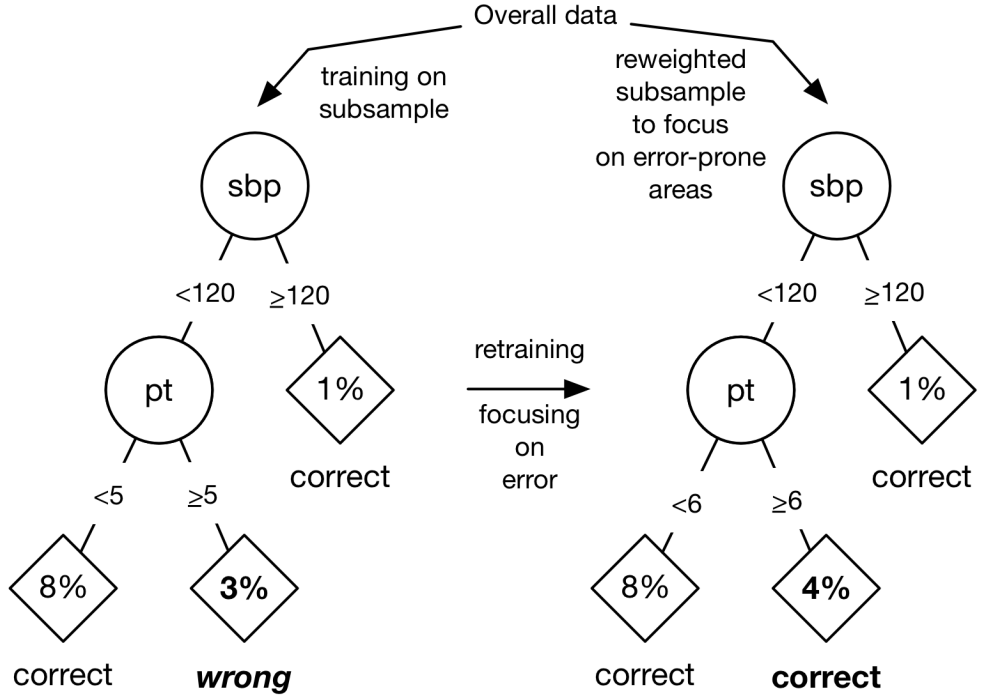
While new machine learning methods are being introduced on a nearly-daily basis, there are a few general types of learners that readers should be familiar with and able to implement.

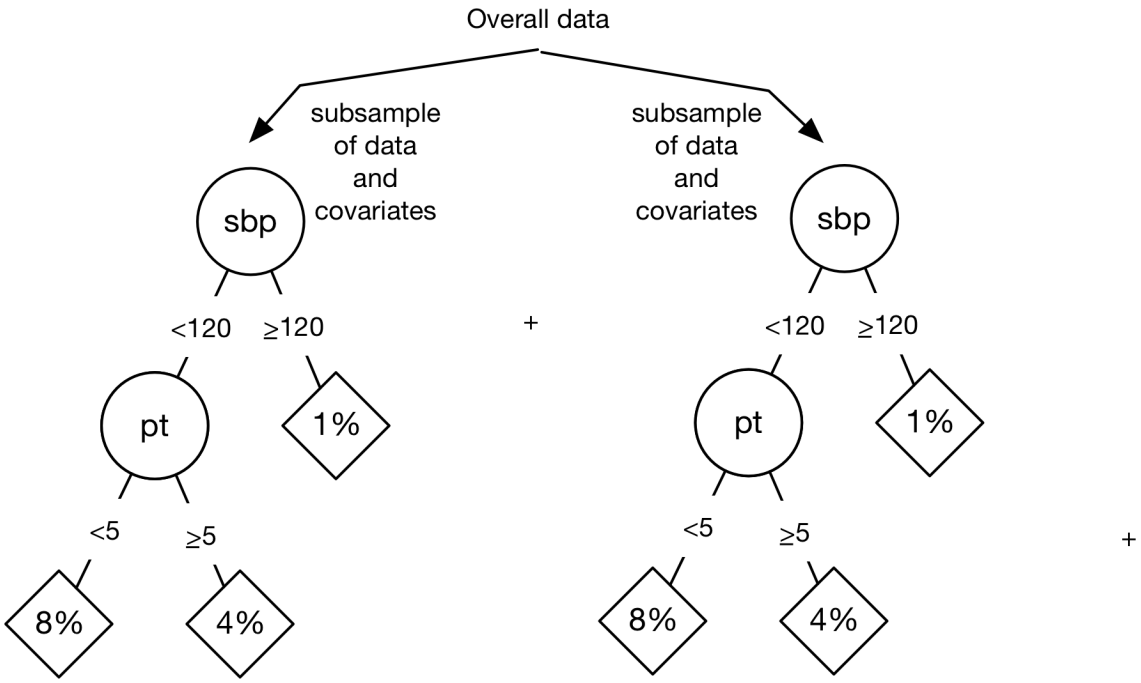
Tree-based learners

The CART analysis we performed earlier in this Chapter was an example of constructing a data-driven decision tree. An advantage of a decision tree is the ability to consider multiple covariates at once, potentially capturing complex interactions between covariates (such as between diabetes and blood pressure) and nonlinearities (since covariates can have different cut-points defining branches, and different predicted outcome probabilities in one branch than in another). However, a limitation of decision trees is that they are prone to overfitting, such that a subgroup may be identified because the decision tree has over-interpreted noise or random outliers in the data as reflecting a real phenomenon or a real subgroup; even cross-validation may not fully correct for the over-fitting. Additionally, just fitting one decision tree may not help fully separate the population into good subpopulations with high between-group variance and low within-group variance, as in our trial data example. We may need many trees, essentially a "forest" of trees, to help select from different combinations of variables and examine how we can best separate out the population for optimal prediction.

Two common methods for constructing a forest, while minimizing the risk of overfitting, are to implement gradient boosting machine (GBM, Figure 9.6) or random forest (RF, Figure 9.7) algorithms. In both GBM and RF algorithms, many trees are grown to subsamples of the training dataset. GBMs average many trees where errors made by the first tree contribute to learning of a more optimal tree in the next iteration (a boosting strategy). RFs average many trees, where each tree is independently fitted with a random subset of covariates selected to be eligible to define the branches (a bagging strategy). Either of these strategies

can be more useful than standard logistic regression when we are trying to predict an outcome that is thought to result from a constellation of complex interacting factors, including multi-level factors such as community-level, household-level, individual-level, and biomolecular-level factors all converging into a persons' risk.





The most common GBM algorithm follows the procedure shown in Equations 9.2 through 9.7 to build k regression trees with incremental boosts of the function f over M iterations:

[Equation 9.2] Initialize $f_{k0} = 0$, k = 1, 2, ..., K

[Equation 9.3] For m = 1 to M, compute the gradient $p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}}$ for all k = 1, 2, ..., K

[Equation 9.4] For k = 1 to K, fit a tree to targets $r_{ikm} = y_{ik} - p_k(x_i)$, for i = 1, 2, ..., N, producing terminal regions R_{jkm}, 1, 2, ..., J_m

[Equation 9.5] Calculate the step $\gamma_{jkm} = \frac{K-1}{K} = \frac{\sum_{x_i \in R_{jkm}} (r_{ikm})}{\sum_{x_i \in R_{jkm}} |r_{ikm}|(1-|r_{ikm}|)}$, j = 1, 2, ..., J_m

[Equation 9.6] Update the function $f_{km}(x) = f_{k,m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jkm} I(\text{x} \in R_{jkm})$

[Equation 9.7] Output $\hat{f}_k(x) = f_{kM}(x)$

The GBM can be sensitive to the choices of the "hyperparameters", which are the learning rate, the maximum tree depth, and column sample rate. The learning rate or shrinkage controls the weighted average prediction of the trees, and refers to the rate (ranging from 0 to 1) at which the GBMs change parameters in response to error (i.e., learn) when building the estimator. Lower learning rates enable slower learning but require more trees to achieve the same level of fit as with a higher learning rate. Lower learning rates also help to avoid overfitting despite being more computationally expensive (i.e., take more computer time and resources to run). The maximum tree depth refers to the number of layers of the three, thus controlling how many splits can occur to separate the population into subpopulations. In the algorithm we demonstrate below, a tree will no longer split if either the maximum depth is reached or a minimum improvement in prediction (relative improvement in squared error reduction for a split $>1^{-5}$) is not satisfied. While deeper trees can provide more accuracy on a derivation dataset, they can also lead to overfitting. The sample rate (ranging from 0 to 1) refer to the fraction of the data sampled among columns of the data for each tree (such that 0.7 refers

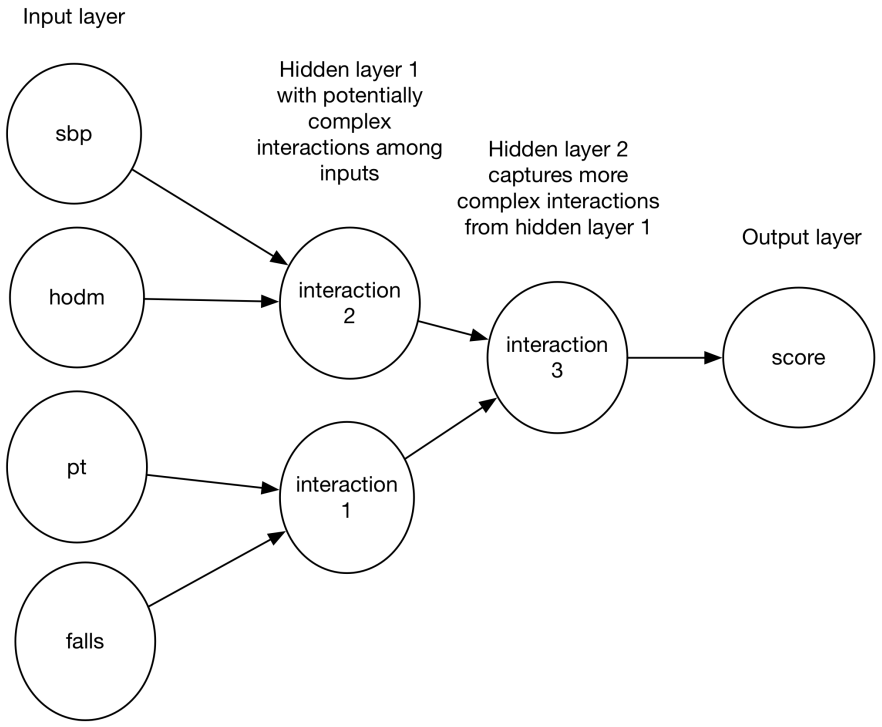
to 70% of the data sampled), which helps improve sampling accuracy. Values less than 1 for the sampling rate can help improve generalization by preventing overfitting. We use a method called "stochastic gradient boosting", in which each GBM iteration uses a subsample of training data drawn at random without replacement.

On the other hand, the RF approach produces a reproducible result with maximum discrimination across a wide range of specifications, thereby not requiring extensive tuning. Hence, the RF approach is often reasonably favored by researchers new to machine learning. In the approach we use, trees are trained in parallel and split until either the maximum depth is reached or a minimum improvement in prediction (relative improvement in squared error reduction for a split $>1^{-5}$) is not satisfied. We can fit forests with varied values of the maximum tree depth, and column sampling rates. Note that random forest utilizes greater maximum tree depth and samples from a smaller number of covariates in each tree than GBM. The greater depth enables random forest to be more efficient on longer datasets, while GBM is more efficient on wider datasets. By randomizing tree building and combining the trees together, random forest helps to reduce overfitting. A simple majority vote of the resulting trees' predictions results in an individual's final prediction of risk from the forest of underlying trees.

Deep learning

Deep learning refers to the training of "neural networks" to predict outcomes; a neural network is a series of data transformations, where the outputs from one series of transformations informs the inputs to the next series of transformations (Figure 9.8). Each transformer (or neuron) in the network takes a weighted combination of inputs reflecting a weighted sum of the observed data (for the first layer of neurons) or from a previous layer of neurons (for the second and subsequent layers of neurons), and produces an output based on a nonlinear transformation function known as an activation function, which can be thought of as analogous to a link function in statistical regression (e.g., like the 'logit' link

that we use in logistic regression). The weights for each sum, and activation function values, determine the output from the network. A standard logistic regression is simply a neural network with a single layer of neurons, in which each transformer multiplies the input covariate by a regression coefficient, and a logistic function is the activation function. Regularization (for instance, LASSO or ridge) and cross-validation techniques are typically applied to neural networks to prevent overfitting.



The simplest deep learning neural network structure, visualized in Figure 9.8, is known as a "feedforward" neural network, in which each layer of neurons is fully connected to the next layer, and information only flows in one direction. By contrast, "recurrent" neural networks have backwards feedback from one layer to a prior layer, which allows for more learning across time series, particularly for problems where knowledge of one prediction affects the next prediction (such as in speech recognition, where the prior word informs the choice of the next word; or in predicting sequential disease events for an individual, where prior diagnoses inform the probability of a subsequent diagnosis). Additional common types of supervised deep learners include: (i) "convolutional"

neural networks, which capture local features of the training dataset, then combine locally-learned networks to gather a global understanding (such as for image recognition, where cluster of pixels form one part of a picture, and these parts are combined to recognize the overall image); (ii) "autoencoders", which take noisy data and try to reconstruct the original data, a process known as "denoising" (also useful for image processing and sound processing); and (iii) "word2vec", which is a series of two-layer neural networks trained to detect words in their context and thereby aid in natural language processing.

At the time of this writing, deep learning neural networks in the medical and public health context remain relatively limited to image recognition (e.g., radiologic detection of abnormalities on X-rays), disease classification problems (e.g., "reading" electronic medical record notes to categorize and classify disease phenotypes in a data-driven manner), and outcome prediction (predicting a clinical outcome based on complex features). We nevertheless demonstrate the application of deep learning to an example dataset, below. Deep learning may be more effective than tree-based methods for outcome prediction when complex context in text (such as in clinical notes or other written assessments) must be processed and included as predictors, when extremely complex interactions are thought to exist between covariates that predict an outcome (such as between multiple -omics markers), or when complex sequential processes must be predicted rather than just a single outcome (e.g., hospital admission, discharge diagnosis, then readmission, and mortality). Deep learning can, however, be difficult to implement because researchers have to choose activation functions, network depths (number of layers), and degree of regularization, among other choices in structuring the model to customize it for the task being accomplished.

The *R* code accompanying this article provides an example of constructing and tuning a standard feedforward neural network, and includes multiple common options for activation functions, network depths, and regularization processes; the code also enables comparison of network learners to tree-based estimators and standard logistic regression for the example problem of predicting

home environmental contamination. Although the statistical code is in *R* to be familiar to epidemiologists and health services researchers, it should be noted that deep learning neural networks are more commonly programmed in Python due to speed and scaling limitations of deep learning in *R*. An excellent online tutorial for more advanced deep learning for medical applications using Python is available at <https://www.deeplearning.ai/>.

The deep learning code we provide implements a multi-layer, feedforward network of neurons, where each neuron takes a weighted combination of input signals as per Equation 9.8:

[Equation 9.8]

$$\alpha = \sum_{i=1}^{n} w_i x_i + b$$

reflecting a weighted sum of the input covariates with additional bias b, which is the neuron's activation threshold. The neuron then produced an output $f(\alpha)$ that represents a nonlinear activation function. In the overall neural network, an input layer matches the covariate space, and is followed by multiple layers of neurons to produce abstractions from the input data, ending with a classification layer to match a discrete set of outcomes. Each layer of neurons provides inputs to the next layer, and the weights and bias values determine the output from the network. Learning involves adapting the weights to minimize the mean squared error between the predicted outcome and the observed outcome across all individuals in the derivation dataset.

In the *R* code accompanying this Chapter, we develop deep learning estimators by training across various activation functions and hidden layer sizes. The activation functions included common maxout, rectifier, and tanh. The maxout activation function is a generalization of the rectified linear function given by Equation 9.9:

[Equation 9.9] $f(\alpha) = max(0, \alpha), f(\bullet) \in \mathbb{R}_+$

In maxout, each neuron chooses the largest output of 2 input channels, where each input channel has a weight and bias value. Users can include a version with "dropout", in which the function is constrained so that each neuron in the network suppresses its activation with probability P (<0.2 for input neurons, and <0.5 for hidden neurons), which scales network weight values towards 0 and forces each training iteration to train a different estimator, enabling exponentially larger numbers of estimators to be averaged as an ensemble to prevent overfitting and improve generalization. The rectifier is a case of maxout where the output of one channel is always 0.

Ensembles

The last decade of machine learning research strongly suggests that while one individual learner, such as one decision tree, can be useful, a constellation of learners can often improve prediction over any single learner, even if the individual learners are each imperfect. Just as a multi-center clinical trial can help us identify generalizable insights better than a single-center trial, or a meta-analysis can help us identify the estimated impact of an intervention better than a single trial, an "ensemble" of learners can help more closely approximate the truth, even when none of the underlying "base learners" is fully correct. Researchers who have little *a priori* theory to favor one type of learner over another can particularly benefit from training an ensemble of learners. To train an ensemble, a researcher will first individually develop learners on the dataset, then use a "meta-learner" (sometimes called a "super learner" or "stacking" method) to combine the predictions of the underlying base learners. The meta-learner gives a weight to each underlying base learner, typically using a strategy such as elastic net regularization to estimate what combination of weights across the base learners minimizes the overall error between the weighted predictions from the base learners and the observed outcomes in repeated samples from the training data. Technically, both GBM and RF are ensemble learners because they

combine individual decision trees to produce a composite prediction. We demonstrate the implementation of ensembles in the accompanying *R* code.

Examples of implementing machine learning in *R*

The book website contains another dataset that we will use to illustrate both tree-based and deep learning methods on a large dataset. The dataset, entitled "09_Basu_sampledata2.csv", is larger than the initial trial dataset, and reflects a simulated cohort study with multiple types of variables. The data can be loaded as with the previous dataset:

```
library(readr)
setwd("~/Downloads")
alldata = read_csv("09_Basu_sampledata2.csv")
View(alldata)
summary(alldata)
```

We see that the data contains information on 100,000 people, each of whom have 100 variables of data that are correlated and inter-dependent in complicated ways. The data includes 40 binary variables, 30 secondary dependent variables correlated with or conditioned upon some of the first 40 variables (reflecting, for example, biomarkers that influence each other), 20 categorical variables, and 10 continuous variables. An unknown subset of these variables predicting the outcome of the disease we wish to predict (a dichotomous outcome for whether or not the person has the condition), with complex interactions and dependencies.

To train machine learners on the data, we first split the data into 'train' and 'test' datasets using the caret package.

```
library(caret)
splitIndex <- createDataPartition(alldata$y, p = .8, list = FALSE, times = 1) #
randomly splitting the data into train and test sets
trainSplit <- alldata[ splitIndex,]
testSplit <- alldata[-splitIndex,]
```

Here, we kept 80% of people in the dataset for training and the remaining 20% of people for testing. Next, we use the h2o package to define the train and test datasets and set the outcome "y" (the disease outcome in the data, a dichotomous 0/1 variable) to be the outcome we want to predict, and all other variables to be the "x" variables we can use for prediction. The h2o.init()command forms a local cluster on your machine to use the various processors on your machine in parallel, for speed. We have to list y as a factor variable for *R* to recognize it properly as a dichotomous variable:

```
library(h2o)
h2o.init()
train <- as.h2o(trainSplit)
test <- as.h2o(testSplit)
y <- "y"
x <- setdiff(names(train), y)
train[,y] <- as.factor(train[,y])
test[,y] <- as.factor(test[,y])
```

In the *R* code on the book website, we first generate a standard logistic regression model using elastic net regularization, for comparison to the more complex machine learning algorithms. The standard logistic regression, as shown in the code, has a C-statistic of 0.67 in the dataset, but with very high error rates in the confusion matrix (of 21%).

For comparison, the next learner we train is a GBM:

```
newgbm = h2o.gbm(x = x, y = y, training_frame = train)
```

For the simple 'newgbm' we created, we can see how well it performs on our held-out test dataset using the h2o.performance command:

```
> h2o.performance(newgbm, data=test)
H2OBinomialMetrics: gbm

MSE:  0.1126128
RMSE:  0.3355782
LogLoss:  0.3768581
Mean Per-Class Error:  0.3623115
AUC:  0.7072977
Gini:  0.4145954

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
```

```
          0    1    Error          Rate
0     14992 1983 0.116819  =1983/16975
1      1838 1186 0.607804   =1838/3024
Totals 16830 3169 0.191060  =3821/19999

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold     value idx
1                        max f1  0.198426 0.383013 188
2                        max f2  0.106251 0.504683 298
3                  max f0point5  0.457326 0.461999 108
4                  max accuracy  0.499738 0.861593  97
5                 max precision  0.823127 1.000000   0
6                    max recall  0.043002 1.000000 382
7                max specificity  0.823127 1.000000   0
8               max absolute_mcc  0.386001 0.318158 115
9   max min_per_class_accuracy  0.133298 0.636937 258
10 max mean_per_class_accuracy  0.154130 0.644040 230
```

The output above shows us that the C-statistic improved over the standard logistic model, to about 0.71 (the 'AUC' metric), and with a lower error rate per the confusion matrix (of 19%). The other output variables are the mean squared error (average squared difference between predicted and observed outcome rates), root-mean-squared-error (square root of the mean squared error), the logarithmic loss (a probability between 0 and 1 of incorrectly classifying people as having high risk), and the mean per-class error (mean error in predicting who gets the disease and who does), for which all values should be closer to zero for better models. The Gini index (not to be confused with the Gini coefficient in economics, which is a measure of inequality) is a measure of 'impurity', or how much the learner has mixed up the groups who get disease from those who don't, and should also be near zero for a perfect model. The performance metrics also include a confusion metric, in which the rows indicate the people predicted to have the disease or not, and how many actually did. The top left and bottom right of the table should be much larger than the top right and bottom left to indicate a good model. The metrics following that are the 'maximum' metrics, but suggest ideally how good the learner can be if a different threshold for the predicted probability (between 0 and 1) is used to define "high risk" of disease. We don't suggest using these metrics, because a single threshold

should be chosen rather than varying the threshold to obtain the best outcome metric. In the *R* code, we show how to obtain alternative metrics and sample across a "grid" of possible values for the learning rate, maximum tree depth, and sampling rates to find the GBM with the highest C-statistic.

Next, we train a random forest, adding in some specifications such as wanting to grow 500 trees and perform 5-fold cross-validation:

```
my_rf <- h2o.randomForest(x = x,
                          y = y,
                          training_frame = train,
                          ntrees = 500,
                          nfolds = 5,
                          keep_cross_validation_predictions = TRUE,
                          seed = 1)
```

The random forest has similar performance to our GBM in terms of the C-statistic (0.69), with a slightly worse confusion matrix:

```
> h2o.performance(my_rf, newdata = test)
H2OBinomialMetrics: drf

MSE:  0.1145182
RMSE:  0.3384054
LogLoss:  0.3835278
Mean Per-Class Error:  0.3647276
AUC:  0.693607
Gini:  0.3872139

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
          0    1    Error          Rate
0     14837 2138 0.125950  =2138/16975
1      1825 1199 0.603505   =1825/3024
Totals 16662 3337 0.198160  =3963/19999
```

Finally, we train a grid of deep learning neural networks, demonstrating difference choices of activation functions, and different LASSO and ridge regularization parameters:

```
activation_opt <- c("Rectifier", "Maxout", "Tanh") # activation function choices
l1_opt <- c(0, 0.00001, 0.0001, 0.001, 0.01) # lasso
l2_opt <- c(0, 0.00001, 0.0001, 0.001, 0.01) # ridge
hyper_params <- list(activation = activation_opt, l1 = l1_opt, l2 = l2_opt)
```

```
search_criteria <- list(strategy = "RandomDiscrete", max_runtime_secs = 300)
dl_grid <- h2o.grid("deeplearning", x = x, y = y,
                    grid_id = "dl_grid",
                    training_frame = train,
                    hyper_params = hyper_params,
                    search_criteria = search_criteria)
dl_gridperf <- h2o.getGrid(grid_id = "dl_grid",
                           sort_by = "AUC",
                           decreasing = TRUE)
```

We find that the best-performing neural network achieves a C-statistic of 0.65 (worse than the logistic regression), which is not necessarily surprising as it's a very complex model for a not-so-complex dataset:

```
> h2o.performance(model = best_dl, newdata = test)
H2OBinomialMetrics: deeplearning

MSE:  0.1221301
RMSE:  0.3494712
LogLoss:  0.4100892
Mean Per-Class Error:  0.3923641
AUC:  0.6482493
Gini:  0.2964986

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
           0    1    Error          Rate
0      14258 2717 0.160059  =2717/16975
1       1889 1135 0.624669   =1889/3024
Totals 16147 3852 0.230312  =4606/19999
```

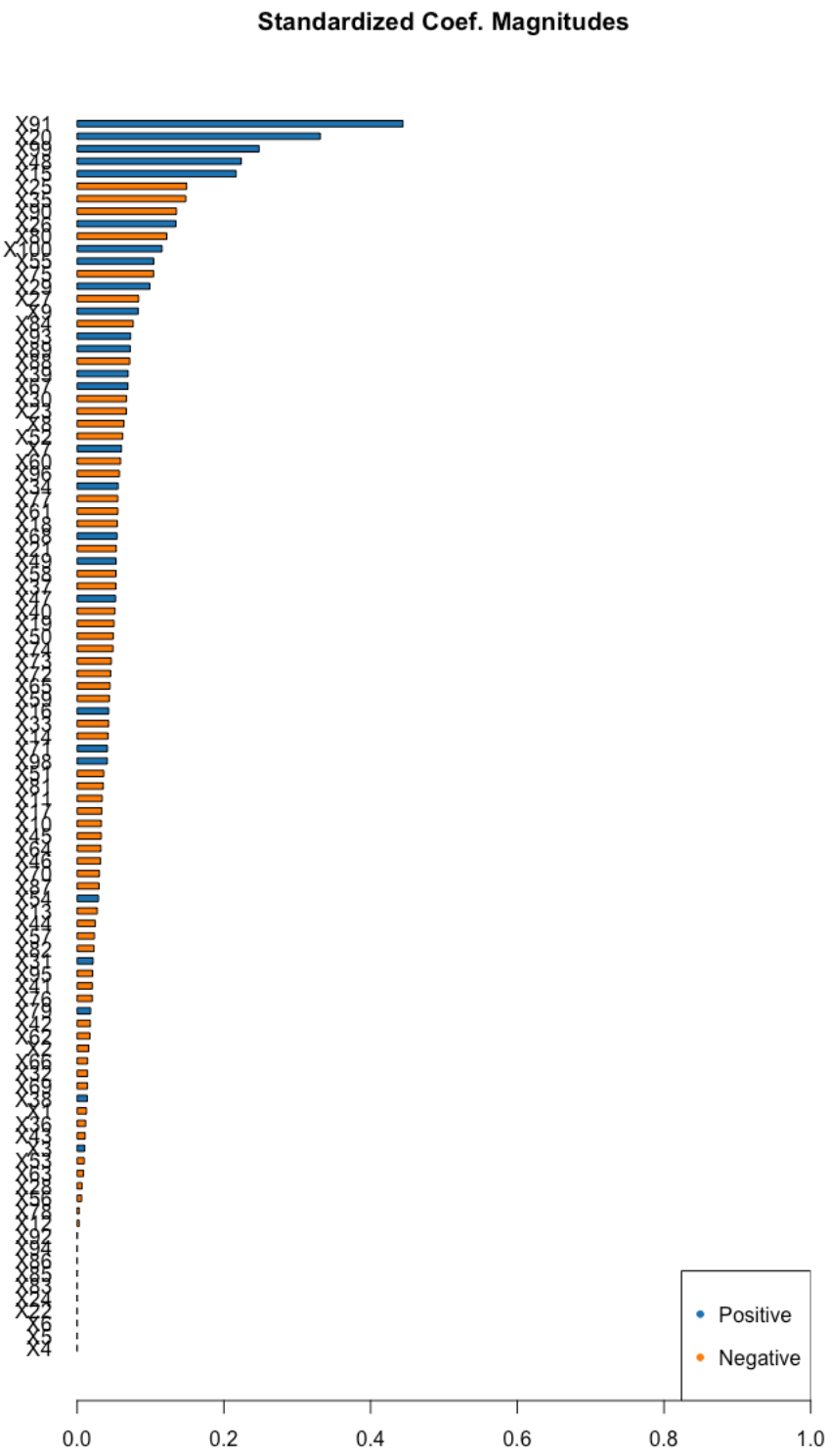
For those readers who are averse to trying multiple learners and tuning across many different variables, the h2o package has a function called h2o.automl, which creates a stacked ensemble of machine learners for those agnostic to any particular model form. The function was created to enable coders to simply designated their dataset and the outcome variable, then allow the package to test a wide range of learners to identify a combination with highest performance per a specified outcome metric (e.g., the C-statistic). The current version of the automl

function includes training and cross-validation of GBM, RF, and neural network grids, along with stacked ensembles of those. The function produces a substantial improvement in both C-statistic (to 0.83) and error rate (down to 16%) in the context of our example problem:

```
>aml <- h2o.automl(x = x, y = y,  training_frame = train, max_runtime_secs =
300)

>aml@leader

MSE:  0.1034877
RMSE:  0.321695
LogLoss:  0.3482245
Mean Per-Class Error:  0.288507
AUC:  0.8324179
Gini:  0.6648357

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
          0     1    Error          Rate
0     48500  5560 0.102849   =5560/54060
1      4616  5119 0.474165    =4616/9735
Totals 53116 10679 0.159511  =10176/63795
```

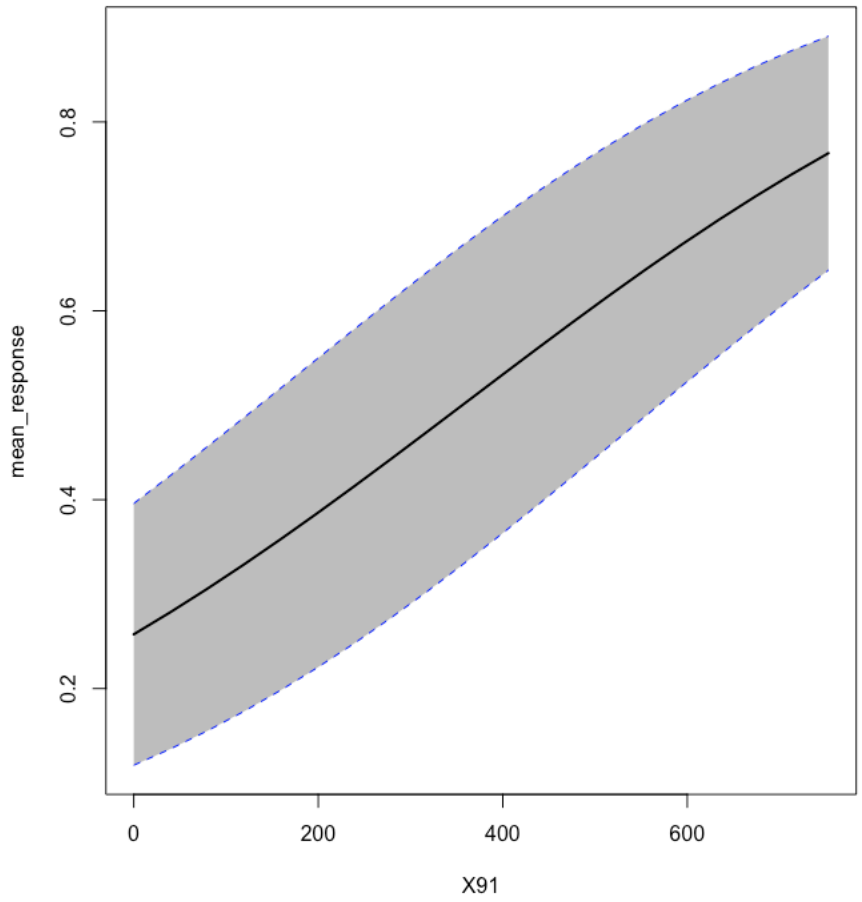
Interpreting machine learners: visualizing inside the black box

At least two key strategies are available to researchers trying to peer inside the "black box" of machine learners, and understand how the learners have interpreted the data to produce useful predictions. The first strategy is known as a "variable importance table" or "variable importance plot", which lists how often different variables are included in the underlying learners, and therefore clarifies which variables are most influential in prediction (Figure 9.9). The plot displays standardized coefficient magnitudes (i.e., coefficients corresponding to how much the outcome probability changes for a standard deviation change in the input variable) across all variables.



The second strategy is known as a "partial dependence plot", which allow us to visualize how learners relate each individual variable to the probability of

the outcome. When covariates go through complex transformations, a partial dependence plot effectively reveals to us how a learner has related values of an input variable with the probability of the outcome variable, including any non-linearities (Figure 9.10).



Machine learning methods can be challenging to learn, but may ultimately provide superior results to traditional regression for some complex research problems at the intersection of precision medicine and health disparities. Here, I reviewed key terms and concepts in machine learning, critical methods for evaluation and interpretation of machine learners, and major common types of learners, with accompanying statistical code to demonstrate their application.

As with many other types of research, machine learning papers are recommended to follow key principles of good research practice. First, to ensure

reproducibility of results, it is essential to publish the code and, if possible, the data that were used to generate the learner. De-identifying data and sharing the raw statistical code is particularly important in the era in which many research claims are found to be suspect. Second, the problem to be answered by a machine learners should be pre-specified, so that researchers are not tempted to use the approach purely to produce (potentially false-positive) associations. Choosing a pre-specified metric for evaluation (e.g., the C-statistic) is an important part of having a pre-specified design for a machine learning project. Third, the end-user of a machine learner must be kept in mind. Different audiences need to either be able to interpret a learner, or just use the learner by inputting data and having the learner automatically provide results (e.g., at the backend of an electronic medical record). Some covariates may be harder to collect than others, and end-users may prefer some metrics (e.g., sensitivity) over others (e.g., specificity). Finally, it is critical that researchers using machine learning methods have "data empathy," or the perspective that the quality and type of data must correspond well to the type of question being asked and the future utilization of the method. If a particular question cannot be answered well with a small dataset, it is unlikely that the question will be better answered with a larger dataset of the same type and data quality. For example, abundant use of insurance claims or electronic medical record data, which are large and widely available in the medical literature, is problematic for clinical studies; prediction models based on such data will not actually predict the presence of disease, but rather the presence of diagnostic billing codes that may poorly correlate to actual disease (and suffer from selection biases and misclassification errors). Hence, it is vital for a researcher to prospectively collect data for a given question, rather than treat machine learning as a "hammer" that can hit any available nail.

Machine learning methods are potentially useful for researchers wanting to classify or predict complex outcomes believed to be influenced by multiple interacting covariates. But often we can do well by starting with a regularized logistic regression model, and rigorously test whether the additional complexity of

a machine learning model is truly justified, or if the additional complexity offers little predictive gain. If prediction is an important and meaningful endeavor for public health and healthcare services research, then producing machine learners that abide by key principles of good research practice may improve the utility, trustworthiness, and beneficial impact of machine learning.