**INTRODUCTION TO ADABOOST OR ADAPTIVE BOOSTING**

**by Simon Moss**

|  |
| --- |
| **Introduction** |

**Example**

 Imagine a researcher who wants to develop an algorithm or app that can predict which research candidates are likely to complete their thesis on time. Specifically, the researcher collates information about 1000 candidates who had enrolled at least 8 years ago and thus should have completed their thesis. An extract of data appears in the following table. Each row corresponds to one individual. The columns represent

* whether the candidates completed their thesis on time
* the grade point average of this candidate during their undergraduate studies—as a percentage of the maximum GPA
* age
* IQ, and
* EQ—or emotional intelligence—as measured by a battery of tests on a 10 point scale

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Complete on time | GPA | Age | IQ | EQ |
| Yes | 85% | 34 | 113 | 7 |
| No | 90% | 27 | 104 | 6 |
| No | 70% | 56 | 142 | 8 |
| Yes | 65% | 71 | 107 | 4 |
| … | … | … | … |  |

 The researcher then utilises these data to construct a model, like a series of formulas, that can be used to predict which applicants in the future are likely to complete their thesis on time. To construct this model, researchers could apply a variety of techniques such as

* logistic regression
* ROC analysis
* the K nearest neighbor algorithm
* decision trees, and
* AdaBoost

This document outlines one of these techniques: AdaBoost, an abbreviation of adaptive boosting. Adaboost is used to predict one of several outcomes, such as whether a research candidate will complete the thesis or not. Compared to many other techniques, AdaBoost

* is a simpler algorithm in which the researcher does not need to reach many decisions to generate an effective model
* has been shown to be effective in many settings—such as when the data comprises few or many predictors
* can generate information that can help researchers interpret or understand the model

**Prerequisites**

 This document, although straightforward, does assume you have developed some preliminary knowledge of decision trees and perhaps random forests. If you have not developed this knowledge, perhaps skim the documents about decision trees and random forests, available on the CDU research website, in the section called “Choosing research methodologies and methods”.

|  |
| --- |
| **Key principles** |

 In essence, AdaBoost applies three principles to classify individuals, animals, or other samples. This section will outline these three principles.

**Many weak learners**

 To predict outcomes, many techniques generate a single model. For example, one technique, called logistic regression, might generate the following equation.

|  |
| --- |
| Loge (odds that a candidate will complete) = .441 x GPA + .007 x IQ - 0.002 x Age + 0.409 x  EQ – 2.668 |

 Similarly, another technique, called decision trees, might generate the following tree or diagram. According to this tree

* if a person reports an IQ over 120, this individual is likely to complete the thesis on time
* if a person does not report an IQ over 120, the computer would then assess EQ
* if the EQ of this person is greater than 5, this individual is likely to complete the thesis on time
* if the EQ of this person is not greater than 5, this individual is unlikely to complete the thesis on time



 In contrast, AdaBoost does not generate one large model. Instead, AdaBoost generates many small models, as illustrated in the following diagram. Indeed

* typically, each model comprises one predictor and two options, sometimes called a stump
* none of these models are especially accurate or informative—and are called weak learners.

But, to predict outcomes, such as whether applicants will complete a thesis or not, AdaBoost combines the predictions of every model to generate a final classification



**Weighting of the weak learners**

 To reiterate, AdaBoost combines predictions from many small models—usually one predictor with two options, called stumps—to generate a single prediction. However, when combining these predictions, AdaBoost does not weight these stumps evenly. Instead

* AdaBoost determines the degree to which each stump generates accurate predictions
* AdaBoost then assigns greater weight to the stumps that generate accurate predictions.

The following example demonstrates this principle. As this display shows

* according to the left diagram or stump, if the IQ of candidates exceeds 120, the model assumes these individuals will complete the thesis
* of the 120 candidates whose IQ did exceed this level, 40 did not complete and are thus errors
* if the IQ of candidates is less than 120, the model assumes these individuals will not complete the thesis
* of the 80 candidates whose IQ was less than 120, 20 did complete and are thus errors
* thus, 60 of the 200 candidates—or 30%--were misclassified
* according to the right diagram or stump, 35% were misclassified



 Therefore, in this example, the left stump generates slightly more accurate predictions that does the right stump. So, to generate the final prediction, the computer would be more influenced by the prediction of the left stump than from the predictions of the right stump. This principle will be clarified soon.

**Gradual improvement of weak learners**

 Thus far, this section has revealed that AdaBoost somehow constructs a series of small models and then weights these models to different extents, depending on their accuracy, when deriving predictions. Importantly, AdaBoost does not generate these small models, or stumps, randomly. Instead

* AdaBoost applies an algorithm to generate the most effective stump possible
* and, unlike other techniques, such as random forests, AdaBoost uses information about previous stumps to gradually improve stumps.

How AdaBoost applies this principle is hard to explain until more detail is presented. The next section will outline these details.

|  |
| --- |
| **Illustration of AdaBoost** |

 The previous section outlined three principles that AdaBoost applies to classify data and predict categorical outcomes. Specifically

* AdaBoost constructs many simple models, called weak learners; these simple models are often stumps
* to generate predictions, AdaBoost somehow weights the contributions of accurate stumps more than inaccurate stumps
* AdaBoost utilizes insights derived from previous stumps to construct additional stumps

This section presents an example that demonstrates how AdaBoost applies these principles. To illustrate, consider again the following data set in which the researcher wants to predict which candidates will complete on time.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Complete on time | GPA | Age | IQ | EQ |
| Yes | 85% | 34 | 113 | 7 |
| No | 90% | 27 | 104 | 6 |
| No | 70% | 56 | 142 | 8 |
| Yes | 65% | 71 | 107 | 4 |
| … | … | … | … |  |

**Assign a weight to each individuals, animal, or sample**

 First, the computer assigns a weight to each row. In this example, each row corresponds to one person. In other datasets, each row might correspond to one animal, specimen, and so forth. This weight, as shown in the final column in the following table, merely equals 1 divided by the number of individuals. At the moment, this number is probably meaningless to you. But, very soon, the importance of this number will become obvious.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Complete on time | GPA | Age | IQ | EQ | **Weight** |
| Yes | 85% | 34 | 113 | 7 | 1/100 or .001 |
| No | 90% | 27 | 104 | 6 | 1/100 or .001 |
| No | 70% | 56 | 142 | 8 | 1/100 or .001 |
| Yes | 65% | 71 | 107 | 4 | 1/100 or .001 |
| … | … | … | … | … | … |

**Identify the best stump**

Next, as illustrated in the following diagram, the computer will evaluate a random stump. The following diagram presents a random stump. In this stump

* individuals whose IQ is greater than 120 are predicted to complete the thesis on time
* individuals whose IQ is less than 120 are predicted to not complete the thesis on time



 The computer, then, evaluates the degree to which this stump classifies the candidates in this dataset accurately. To achieve this goal, the computer, in principle, generates something like the following table. Specifically

* the final column now indicates whether the stem classified this person correctly
* to illustrate, the first participant completed on time
* but, because the IQ of this person is less than 120, the stem would predict this person will not complete on time
* this person will thus be classified incorrectly; the prediction is inaccurate.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Complete on time | GPA | Age | IQ | EQ | **Weight** | **Incorrect** |
| Yes | 85% | 34 | 113 | 7 | 1/100 | 1 |
| No | 90% | 27 | 104 | 6 | 1/100 | 0 |
| No | 70% | 56 | 142 | 8 | 1/100 | 1 |
| Yes | 65% | 71 | 107 | 4 | 1/100 | 1 |
| … | … | … | … | … | … |  |

 So, to evaluate the stem, the computer adds all the weights that correspond to incorrect predictions—such as 0.01+ 0.01 + 0.01 and so forth. Note that

* high sums indicate inaccurate models
* low sums indicate accurate models.

The computer then applies this procedure to other stumps, such as stumps in which candidates are predicted to complete the thesis if

* IQ exceeds 1000
* GPA exceeds 5
* EQ exceeds 4, and so forth

Ultimately, the computer will choose the stump that generates the best prediction, as represented by the lowest sum. In practice, the computer does not attempt every possible threshold. Instead, the computer applies a more systematic approach. But regardless, the outcome is the computer will identify the stump that classifies the candidates most accurately.

**Prioritize this stem**

The computer then computes an interesting number. To appreciate this number

* remember that, ultimately, AdaBoost generates many stems
* AdaBoost then integrates the predictions of all these stems to generate a final prediction
* when generating this final prediction, AdaBoost prioritizes some stems more than other stems

This number represents the extent to which AdaBoost will prioritize each stem. To calculate this number, the computer utilises the following equation. You do not need to understand this equation. You should merely realise this priority is higher when the stem misclassifies fewer individuals

|  |
| --- |
| Priority assigned to this stem = ½ log ((1 – total error)/ (total error))NB: total error refers to the number of misclassifications |

**Update the weights**

The computer is now ready to initiate a second round of calculations. To start this second round, the computer adjusts the weights. To adjust the weights, the computer applies the following formula

|  |
| --- |
| If the row or individual had been misclassified in the previous round* Updated weight = Previous weight x epriority assigned to the previous stem

If the row or individual had been classified correctly in the previous round* Updated weight = Previous weight x e -priority assigned to the previous stem
 |

This formula is hard to understand. But, in essence, this formula merely

* increases the weight assigned to rows or individuals that had been misclassified, especially if the stem had been accurate overall
* decreases the weight assigned to rows or individuals that had been classified correctly, especially if the stem had been accurate overall

For example, as the following table reveals, the weights associated with rows that had been classified incorrectly are now larger than .01: the previous weight. The weights associated with rows that had been classified correctly are now smaller than .01.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Complete on time | GPA | Age | IQ | EQ | **Weight** | **Incorrect** |
| Yes | 85% | 34 | 113 | 7 | .06 | 1 |
| No | 90% | 27 | 104 | 6 | .004 | 0 |
| No | 70% | 56 | 142 | 8 | .06 | 1 |
| Yes | 65% | 71 | 107 | 4 | .06 | 1 |
| … | … | … | … | … | … |  |

Why does this formula increase the weight assigned to rows or individuals that had been misclassified? What is the purpose of these changes? The reason is clarified next.

**Identify the best stump again**

AdaBoost now continues this sequence of phases many times, called rounds. That is, on each round, AdaBoost

* identifies the best stump—that is, the stump that minimizes the sum of weights that corresponds to incorrect classifications
* assigns a priority to this stump
* updates the weights assigned to each row or individual in the datafile

Because the weights change over time, each round will generate a distinct stump. For example,

* during the second round, the stump might divide individuals depending on whether their EQ exceeds 5. That is, this stump might minimize the sum of weights that correspond to incorrect classifications.
* during the third round the stump might divide individuals depending on whether their GPA exceeds 75%. That is, this stump might minimize the sum of weights that correspond to incorrect classifications and so forth

More importantly, the weights are increased in the rows that were misclassified in the previous round. Therefore, during each round, the chosen stump is more likely to predict the individuals who had been misclassified in the previous round accurately.

**Classify additional individuals or cases**

 Finally, after constructing a series of stems, AdaBoost can then classify additional individuals or cases—individuals or cases that were not used to generate the stems. To illustrate,

* suppose an applicant reports a GPA of 75, an IQ of 120, an age of 50, and an EQ of 6
* these data and then subjected to each stem
* as the following figure shows, each stem will then predict whether the person will complete the thesis or not
* in this instance, assume that most of the stems predict the person will complete the thesis
* hence, the algorithm would predict this person will complete the thesis



 The algorithm does not, however, merely determine the proportion of stems that predict the person will complete the thesis. Instead, the algorithm somehow prioritizes some of the stems over other stems—using the priorities that were calculated before. To illustrate, consider the following table:

* each row corresponds to a distinct stump
* the first column specifies the round in which this stump was constructed
* the second column specifies the predicted outcome; 0 indicates the individual is unlikely to complete the thesis on time and 1 indicates the individual is likely to complete
* the third column specifies the priority that was assigned to this stump
* the fourth column is the product of the second and third column.
* the final number, .20, is the sum of products over the sum of priorities

|  |  |  |  |
| --- | --- | --- | --- |
| Round | Prediction | Priority | Product |
| 1 | 1 | .24 | .24 |
| 2 | 0 | .14 | 0 |
| 3 | 1 | .12 | .12 |
| … | … | … |  |
|  |  |  | Sum of productsSum of priority  |
|  |  |  | = .20 |

 If the prediction the person will complete is frequent and prioritised, this final value will tend to exceed 0.5. If the prediction the person will not complete is frequent and prioritised, this final value will tend to be less than 0.5. Therefore, in this example, because the value is appreciably lower than 0.5, the algorithm will predict the applicant will not complete the thesis on time.

|  |
| --- |
| **How to conduct AdaBoost. Step 1: Install and use R** |

You can use a variety of statistical packages to apply this technique. This document will show you how to conduct this technique in software called R. If you have not used R before, you can download and install this software at no cost. To achieve this goal

* visit <https://www.cdu1prdweb1.cdu.edu.au/files/2020-08/Introduction%20to%20R.docx> to download an introduction to R
* read the section called Download R and R studio
* although not essential, you could also skim a few of the other sections of this document to familiarize yourself with R.

|  |
| --- |
| **How to conduct AdaBoost. Step 2: Upload the data file** |

Your next step is to upload the data into R. To achieve this goal

* open Microsoft Excel
* enter your data into Excel; you might need to copy your data from another format. Or your data might already have been entered into Excel

In particular, as the following example shows

* each column should correspond to one variable
* each row should correspond to one individual, animal, specimen, and so forth
* the first row labels the variables
* to prevent complications, use labels that comprise only lowercase letters—although you could end the label with a number, such as age3
* in the first column, 0s represent candidates who did not complete the thesis; 1 represent candidates who did complete the thesis



To convert this file into a csv file—such as a file called research.data—and then to upload this file into R studio

* visit <https://www.cdu1prdweb1.cdu.edu.au/files/2020-08/Introduction%20to%20R.docx> to download the introduction to R—unless you have already downloaded this document
* read the section called “Upload some data”

|  |
| --- |
| **How to conduct AdaBoost. Step 3: Enter the code and interpret the results** |

 To conduct AdaBoost, you need to enter some code. The code might resemble the following display. At first glance, this code looks absolutely terrifying. But actually, this code is straightforward once explained.

|  |
| --- |
| install.packages("adabag")install.packages("caret")library(adabag)library(caret)indexes=createDataPartition(research.data$completion, p=.75, list = F)train = research.data[indexes, ]test = research.data[-indexes, ]train[["completion"]] = factor(train[["completion"]]) test[["completion"]] = factor(test[["completion"]])model = boosting(completion~., data=train, boos=TRUE, mfinal=50)print(names(model))print(model$trees[1])pred = predict(model, test)print(pred$confusion)print(pred$error)result = data.frame(test$completion, pred$prob, pred$class)print(result) |

 To enter code, you could write one row, called a command, at a time in the Console. But, if you want to enter code more efficiently,

* in R studio, choose the File menu and then “New File” as well as “R script”
* in the file that opens, paste the code that appears in the left column of the following table
* to execute this code, highlight all the instructions and press the “Run” button—a button that appears at the top of this file

You should not change the bold characters in the left column. You might change the other characters, depending on the name of your data file, the name of your variables, and so forth. The right column of the following table explains this code. You do not, however, need to understand all the code.

|  |  |
| --- | --- |
| Code to enter | Explanation or clarification |
| **install.packages("adabag")****install.packages("caret")****library(adabag)****library(caret)** | * R comprises many distinct sets of formulas or procedures, each called a package
* *adabag* and *caret* are two packages and can be used to conduct AdaBoost
* *install.packages* merely installs this package onto the computer
* *library* then activates this package
* the quotation marks should perhaps be written in R rather than Word; the reason is that R recognises this simple format— " —but not the more elaborate format that often appears in Word, such as “ or ”.
 |
| **set.seed**(3033) | * Later, the computer will be asked to identify some random numbers
* This code, however, instructs the computer to begin these random numbers at position 3033
* Consequently, you could, if you wanted, identify the same random numbers again
 |
| **indexes=createDataPartition (**research.data$completion, **p=**.75**, list = F)** | * This code randomly selects 0.75 or 75% of the rows in the data file *research.data*—or at least 75% of the rows that contain data in the column called *completion*
* These rows are then stored in a container or variable called *indexes*
* For example, *indexes* might contain numbers like 1, 3, 4, 6, 9, 10, 13, and 16—representing 75% of the rows
 |
| **train =** research.data**[indexes, ]** | * This code extracts all the rows in the data file called *research.data* that correspond to *indexes*—such as rows 1, 3, 4, 6, 9, 10, 13, and 16
* This subset of rows is stored in a data file or container called *train*
* Thus *train* represents the training data—the data that will be used to construct the stumps or model
 |
| **test =** research.data**[-indexes, ]** | * This code extracts all the rows in the data file called *research.data* that do not correspond to *indexes*—such as rows 2, 5, 7, 8, 11, and 12
* This subset of rows is stored in a data file or container called *test*
* Thus *test* represents the testing data—the data that will be used to test whether the predictions are accurate

This distinction between training data and testing data is common in machine learning. That is, researchers often* use a subset of data, called training data, to construct a model—in this instance, a set of stumps
* use the remaining data, called testing data, to assess whether this model can correctly classify individuals who were not used to construct this model
 |
| train[["completion"]] = **factor**(train[["completion"]]) test[["completion"]] = **factor**(test[["completion"]]) | * R might assume that *completion—*a column that contains 1s and 0s—is a numerical variable
* But, to undertake AdaBoost, R must recognize these 1s and 0s actually represent categories, such as individuals who completed their thesis and individuals who did not complete their thesis
* This code merely converts the column labelled completion to a categorical variable—sometimes called a factor.
 |
| **model = boosting(**completion~., **data=train, boos=TRUE,** mfinal=50**)** | * This code actually constructs all the stumps and thus develops the model
* The code *completion~.,* informs R that completion is the outcome variable and every other column in the data file are the predictors
* The code *data=train* instructs R to use the data in the container called train to construct this model
* The code *mfinal=50* instructs R to develop 50 stumps. If you increase this number, R will develop a more accurate model—but the procedure might be delayed.
 |
| **print(model$trees[1])** | * This code prints various properties about the model—but these properties demand significant expertise to interpret
 |
| **pred = predict(model, test)** | * This code applies the model to the test data
* In this example, this model predicts whether the individuals in the test data will complete the thesis or not
 |
| **print(pred$confusion)** | * This code is designed to construct what is called a confusion matrix, as shown below
* To illustrate, the 3 in this table indicates that 3 individuals in the test data did not complete the thesis but were predicted to complete
* That is, the observed or actual category is 0; the predicted category is 1
* The 4 indicates that that 4 individuals in the test data did complete the thesis and were predicted to complete
* This table might demand a few seconds to understand

 Observed ClassPredicted Class 0 1 0 1 0 1 3 4 |
| **print(pred$error)** | * Reports the proportion of individuals in the test sample that were classified incorrectly
* You might then compare this proportion with the proportion that other techniques generate—such as support vector machines.
* In your report, you would indicate which technique generated the least errors.
 |
| **result = data.frame(test$completion, pred$prob, pred$class)****print(result)** | * This code orints the probability that each individual in the test data belongs to each class—such as the individuals who completed the thesis and the individuals who did not complete the thesis
* In this example, recall that 0 is the first category, representing individuals who did not complete the thesis.
* So X1 corresponds to the individuals who did not complete the thesis.
* X2 corresponds to the individuals who did complete the thesis.
* The probability the first person belongs to X2 is greater than 0.5. Hence, this person was classified as X2 or likely to complete.

 test.completion X1 X2 pred.class1 0 0.4014547 0.5985453 12 0 0.4927353 0.5072647 13 1 0.4019896 0.5980104 14 1 0.2900161 0.7099839 15 1 0.4180466 0.5819534 16 0 0.4065547 0.5934453 17 0 0.7135936 0.2864064 08 1 0.3199136 0.6800864 1 |