The Random Forest Algorithm for Statistical Learning with Applications in Stata

Rosie Yuyan Zou, Matthias Schonlau

August 2018

1 Abstract

Random Forest is a statistical or machine learning algorithm for prediction. We introduce a corresponding new Stata command, `randomforest`. We give an overview of the random forest algorithm and illustrate its use with two examples. The first example is a classification problem that predicts whether a credit card holder will default on his or her debt; the second example is a regression problem that predicts log-scaled number of shares of online news articles. We conclude with a discussion that summarizes key points demonstrated in the examples.

2 Introduction

In recent years the use of statistical or machine learning algorithms has increased in the social sciences.[1] For instance, to predict economic recession, Nyman and Ormerod (2017) compared ordinary least squares regression results with random forest regression results and obtained a considerably higher adjusted R-squared value with random forest regression (Nyman & Ormerod, 2017) as compared to ordinary least squares regression. In economics, a recent book (Basuchoudhary, Bang, & Sen, 2017) gives an overview over various machine learning algorithms for predicting economic growth and recession. In environmental science, a recent paper used learning algorithms including LASSO regression, Random Forests and neural networks to predict ragweed pollen concentration based on 27 years of historical data and a total of 85 predictor variables. The best predictive performance was obtained using random forests (Liu et al., 2017).

When prediction is the main goal, statistical learning algorithms are typically preferable to linear regression. Such algorithms are well suited for medium to large data sets and even when there are a large number of variables relative to the number of observations. Linear and logistic regression do not run when the number of variables is larger than the number of observations.

Random Forests is one of the best performing learning algorithms. In both Nyman and Ormerod's (2017) and Liu's (2017) examples, random forest has proven to be the optimal method in terms of prediction accuracy. For social scientists such developments in algorithms are only useful to the extent that they can access an implementation of the algorithm. This paper introduces a Stata command for random forests developed by the authors that is built on the WEKA library (Frank, Hall, & Witten, 2016; Hall et al., 2009).

[1] Statistical learning and machine learning are synonymous. We use statistical learning for the remainder of the article.

The outline of this paper is as follows: In Section 3, we briefly discuss the Random Forest algorithm. Section 5 gives an example for predicting whether a given credit card user will default on his or her debt. Section 6 gives an example for estimating log-scaled number of shares of online news articles. Section 7 concludes with a discussion.

3 The Random Forest algorithm

We first discuss tree models because they form the building blocks of the Random Forest algorithm. A tree-based model involves recursively partitioning the given data set into two groups based on a certain criterion until a predetermined stopping condition is met. At the bottom of decision trees are so-called leaf nodes or leaves.

Figure 1 illustrates a recursive partitioning of a 2-dimensional input space with axis-aligned boundaries – i.e. each time the input space is partitioned in a direction parallel to one of the axes. Here, the first split occurred on $x_2 >= a_2$. The two subspaces where then again partitioned: The left branch was split on $x_1 >= a_4$. The right branch was first split on $x_1 >= a_1$ and one of its subbranches was split on $x_2 > a_3$. Figure 2 is a graphical representation of the subspaces partitioned in Figure 1. Depending on how the partition and stopping criteria are set, decision trees can be

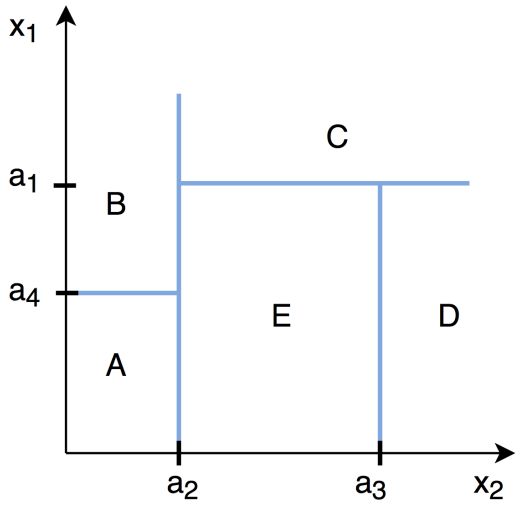

Figure 1: Recursive Binary Partition of a 2-Dimensional Subspaces

designed both for classification tasks (categorical outcome, e.g. logistic regression) and regression tasks (continuous outcome).

For both classification and regression problems, the subset of predictor variables selected to split an internal node depends on predetermined splitting criteria which is formulated as an optimization problem. A common splitting criterion in classification problems is entropy, which is the practical application of Shannon's source coding theorem that specifies the lower bound on the length of a random variable's bit representation (Shannon, 2001). At each internal node of the decision tree,

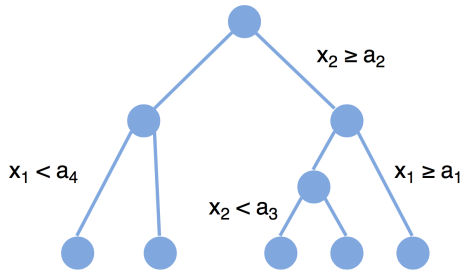

Figure 2: A Graphical Representation of the Decision Tree in Figure 1

entropy is given by the formula

$$E = -\sum_{i=1}^{c} p_i \times log(p_i)$$

where c is the number of unique classes and p_i is the prior probability of each given class. This value is maximized in order to gain the most information at every split of the decision tree. For regression problems, a commonly used splitting criterion is the mean squared error at each internal node.

A drawback of decision trees is that they are prone to over-fitting, which means that the model becomes too "catered" towards the training data set and performs poorly on a new data set – i.e. the test data. Over-fitting decision trees will lead to low general predictive accuracy, herein referred to as generalization accuracy.

One way to increase generalization accuracy is to only consider a subset of the samples and build many individual trees. First introduced by Ho, this idea of the random subspace method was later extended and formally presented as Random Forests by Breiman (Ho, 1995; Breiman, 2001). The Random Forest model is an ensemble tree-based learning algorithm; that is, the algorithms averages predictions over many individual trees. The algorithm also utilizes **bootstrap aggregating**, also known as **bagging**, to reduce over-fitting and improve generalization accuracy. Bagging refers to fitting each tree on a bootstrap sample rather than on the original sample. The algorithm is as follows:

```
for i ← 1 to B do
    Draw a bootstrap sample of size N from the training data;
    while node size != minimum node size do
        randomly select a subset of m predictor variables from total p;
        for j ← 1 to m do
            if j-th predictor optimizes splitting criterion then
                split internal node into two child nodes;
                break;
            end
        end
    end
end
return the ensemble tree of all B sub-trees generated in the outer for loop;
```

Algorithm 1: Random Forest Algorithm

The random forest algorithm gives a more accurate estimate of the error rate, as compared with decision trees. More specifically, the error rate has been mathematically proven to always converge (Breiman, 2001). In a classification problem, the error rate is approximated by the **out-of-bag error** during the training process. In each tree of the random forest, the out-of-bag error is calculated based on predictions for observations that were not in the bootstrap sample for that tree. Finding parameters that would produce a low out-of-bag error is often a key consideration in model selection and parameter tuning. Note that in the random forest algorithm, the size of the subset of predictor variables, m, is crucial to controlling the final depth of the trees. Hence it is a parameter that needs to be tuned during model selection, which will be discussed in the examples later.

In order to gain some insight in the complex model, the so-called importance of each variable is calculated. Variable importance is calculated by adding up the improvement in the objective function given in the splitting criterion over all internal nodes of a tree and across all trees in the forest, separately for each predictor variable. In the Stata implementation of random forest, the variable importance score is normalized by dividing all scores over the maximum score: The importance of the most importance variable is always 100%.

4 Stata Syntax

The Stata syntax to fit a random forest model is:

```
randomforest depvar indepvars [if] [in] , [ options ]
```

with the following post-estimation command:

```
predict newvar | varlist | stub* [if] [in] , [ pr ]
```

5 Example: Credit Card Default

The following sample data comes from a 2009 research project on data mining techniques for the predictive accuracy of probability of default of credit card clients (Yeh & Lien, 2009; Dheeru & Karra Taniskidou, 2017).[2] There are a total of 30,000 observations, 1 response variable, 22 explanatory variables, and no missing values. The response variable is a binary variable that encodes

[2]To access the exact data set used in this example, please visit
`https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients`

whether the card holder will default on his/her debt, with 0 encoded as "no default" and 1 encoded as "default". 10 of the 22 explanatory variables are categorical variables containing information such as gender, education, marital status, and whether past payments have been made on time or delayed. The remaining 12 continuous explanatory variables contain information on the monthly bill amount and payment amount over six months. For a complete list of variables, please refer to Appendix A.

In this example we will investigate what are the predominant factors that affect credit card default prediction accuracy, as well as contrast the prediction accuracy obtained using both random forest and logistic regression.

5.1 Model Training and Parameter Tuning

To start the model training process, the data points are arranged in randomly sorted order. When the data are split into training and test data, random sort order ensures that the training data are random as well. To allow for reproducible results, a seed value is set. Then the data set is split into two subsets: 50% used for training and 50% used for testing (validation). In small data sets a 50-50 split may reduce the size of the training data too much; for this relatively large data set a 50-50 split is not problematic. The randomization process mentioned previously ensures that the training data contains observations belonging to all available classes, so as long as the class probabilities are not heavily imbalanced. Additionally, it removes the model's potential dependency on the ordering of observations relative to the test data. Finally, since the variable for marital status uses values $0, 1, 2, 3$ to encode un-ordered categorical information, we need to create 4 new binary indicator variables for each marital status using the command `tab marriage, gen(marriage_enum)`. Creating the fourth indicator variable is redundant, but this does not matter to tree-based algorithms like random forest.

Next, the hyper parameters, parameter values that we need to determine prior to model building, are tuned to find the model with the highest testing accuracy. Specifically, the number of iterations (i.e. number of sub-trees) and number of variables to randomly investigate at each split, *numvars*, are tuned. The following code segment iteratively calculates the out-of-bag prediction accuracy as a function of the number of iterations and *numvars*. The number of iterations starts at 10 and is incremented by 5 every time until it reaches 500. We will use both OOB Error (tested against training data subsets that are not included in sub-tree construction) and validation error (tested against the testing data) in order to determine the best possible model.

Usually tuning parameters requires grid searching, i.e. exhaustively searching through a user-specified subspace of hyper parameter values. In this case, however, since random forest OOB error rates converge after the number of iterations get large enough, we simply need to set the iterations to a value large enough for convergence to have occurred prior to tuning the *numvars* parameter.

To illustrate how the OOB error and validation error have similar trends as the number of iterations grow, the random forest function is iteratively called. The variable *iteration* is initialized to 10 and increments by 5 per function call until it reaches 500. Finally, the trends of OOB error and validation error can be visualized by plotting those values against the number of iterations, as shown in Figure 3.

```
use default.dta
set seed 201807
gen u=uniform()
sort u
```

```
// figure out how large the value of iterations need to be
gen out_of_bag_error1 = .
gen validation_error = .
gen iter1 = .
local j = 0
forvalues i = 10(5)500{
local j = `j' + 1
randomforest defaultpaymentnextmonth limit_bal sex ///
education marriage_enum* age pay* bill* in 1/15000, type(class) iter(`i') numvars(1)
replace iter1 = `i' in `j'
replace out_of_bag_error1 = `e(OOB_Error)' in `j'
 predict p in 15001/30000
 replace validation_error = `e(error_rate)' in `j'
 drop p
}

label var out_of_bag_error1 "Out of Bag Error"
label var iter1 "Iterations"
label var validation_error "Validation Error"
scatter out_of_bag_error1 iter1, mcolor(blue) msize(tiny) || ///
scatter validation_error iter1, mcolor(red) msize(tiny)
```

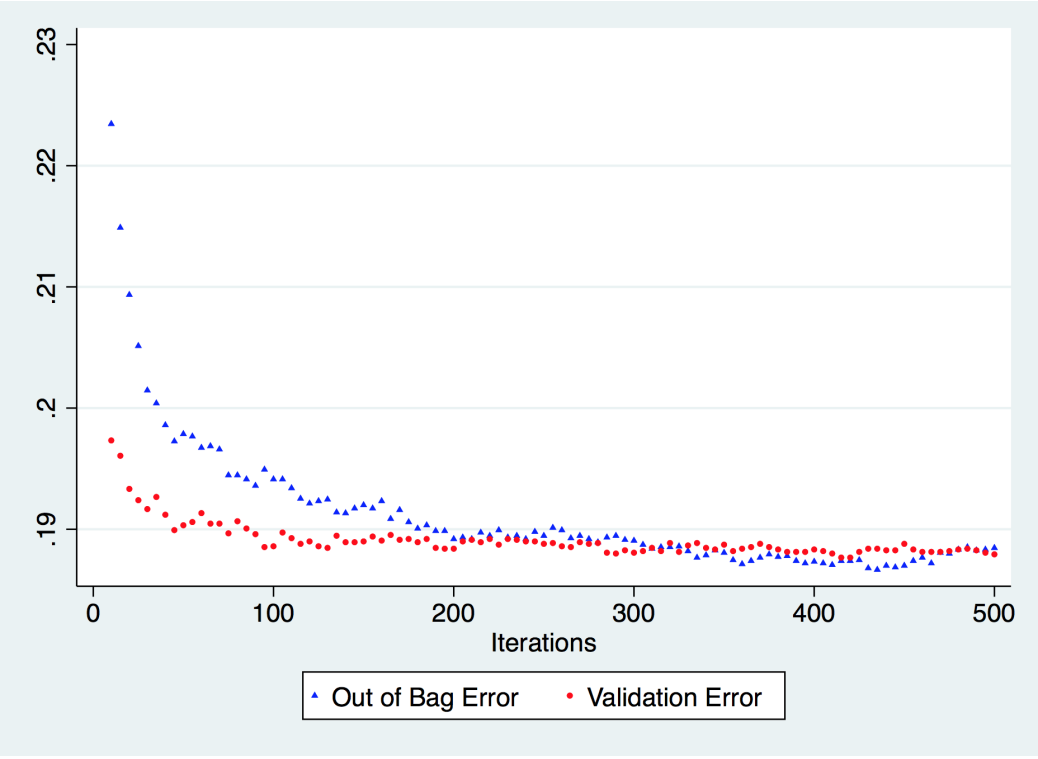

Figure 3: Out of Bag Error and Validation Error vs. Iterations Plot

We can see from Figure 3 generated by the above code block that both the OOB error and the

validation error stabilize around 19% for iterations equal to 500, with a slightly decreasing trend. Hence fixing the number of iterations at 500 is a good choice.

Next we can tune the hyper parameter *numvars*:

```
gen oob_error = .
gen nvars = .
gen val_error = .
local j = 0
forvalues i = 1(1)26{
local j = `j' + 1
randomforest defaultpaymentnextmonth limit_bal sex ///
education marriage age pay* bill* in 1/15000, type(class) iter(500) numvars(`i')
 replace nvars = `i' in `j'
replace oob_error = `e(OOB_Error)' in `j'
predict p in 15001/30000
replace val_error = `e(error_rate)' in `j'
drop p
}
label var oob_error "Out of Bag Error"
label var val_error "Validation Error"
label var nvars "Number of Variables Randomly Selected at Each Split"
scatter oob_error nvars, mcolor(blue) msize(tiny) || ///
scatter val_error nvars, mcolor(red) msize(tiny)
```

Using the commands `sum val_error` and `list val_error`, we can see that at $numvars = 14$, we get the lowest validation error at .1824667. Hence we will use $numvars = 14$ for our final model.

In principle the random forest can output an OOB error at each iteration. However, the WEKA implementation of random forest used for the Stata plugin does not output running calculations of OOB error as the algorithm runs and instead only outputs one final OOB error for the total number of iterations. This means that tuning the iterations parameter requires running the random forest algorithm k times for every value of $iterations = k$. In order to make this process efficient, it is best to set min and max values and a reasonable increment for us to be able to see the trend of the change of OOB error over increasing iterations.

5.2 Final Model and Interpretation of Results

As shown in the previous section, we have set the values of hyper-parameters to be `iterations = 500` and `numvars = 14`. The following code block gives the final model and prediction error:

```
// final model: numvars = 14, iter = 1000
randomforest defaultpaymentnextmonth limit_bal sex ///
education marriage age pay* bill* in 1/15000, type(class) iter(1000) numvars(14)
ereturn list

predict prf in 15001/30000
ereturn list
```

The final out-of-bag error is 18.33%, slightly larger than the actual prediction error, which is 18.01%, calculated over 15,000 test observations. We can see from both Figure 1 and Figure 2 that the out-of-bag error and validation error have the same pattern when plotted against the two hyper parameters,

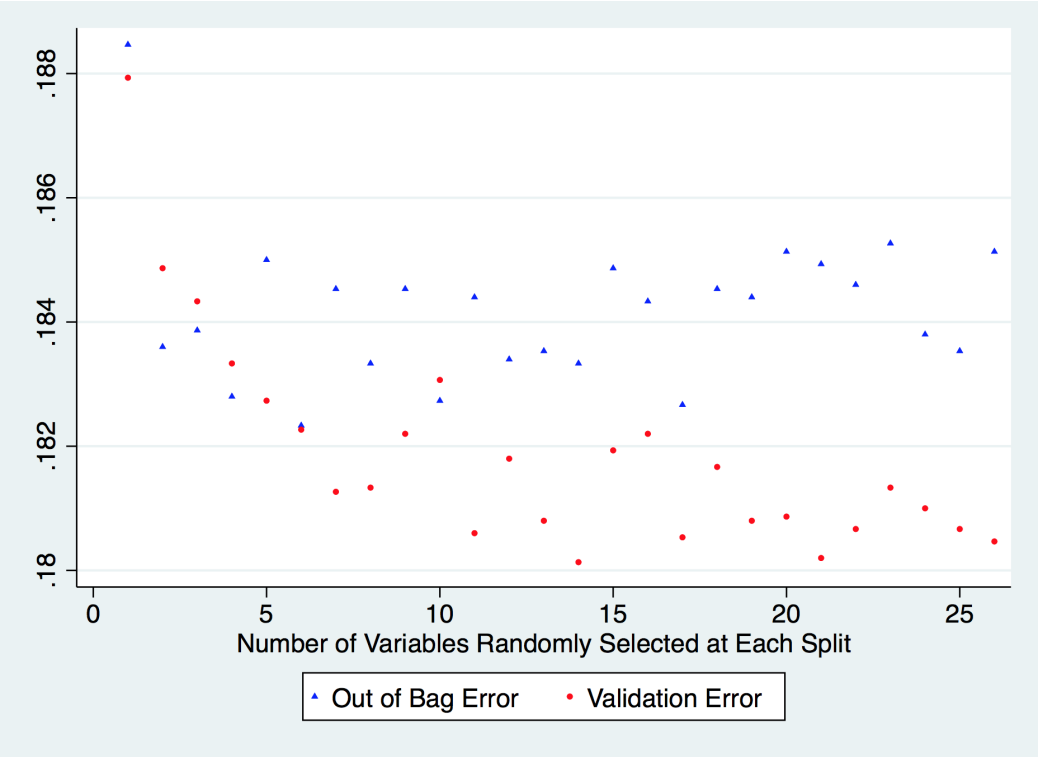

Figure 4: Out of Bag Error and Validation Error vs. Number of Variables Plot

iterations and number of variables. In practice, random forest out-of-bag errors reliably estimates the actual error.

We also would like to ascertain which factors are most important in the prediction process. Random forests are black-boxes in that they don't offer insight in how the predictions is accomplished. Variable importance scores of each predictor provide some limited insight. The following code segment plots the variable importance:

```
// variable importance plot
matrix importance = e(importance)
svmat importance
gen importid=""

local mynames : rownames importance
local k : word count `mynames'
if `k'>_N {
   set obs `k'
}
forvalues i = 1(1)`k' {
    local aword : word `i' of `mynames'
    local alabel : variable label `aword'
    if ("`alabel'"!="") qui replace importid= "`alabel'" in `i'
    else qui replace importid= "`aword'" in `i'
}
```

```
graph hbar (mean) importance, over(importid, sort(1) label(labsize(2))) ytitle(Importance)
```

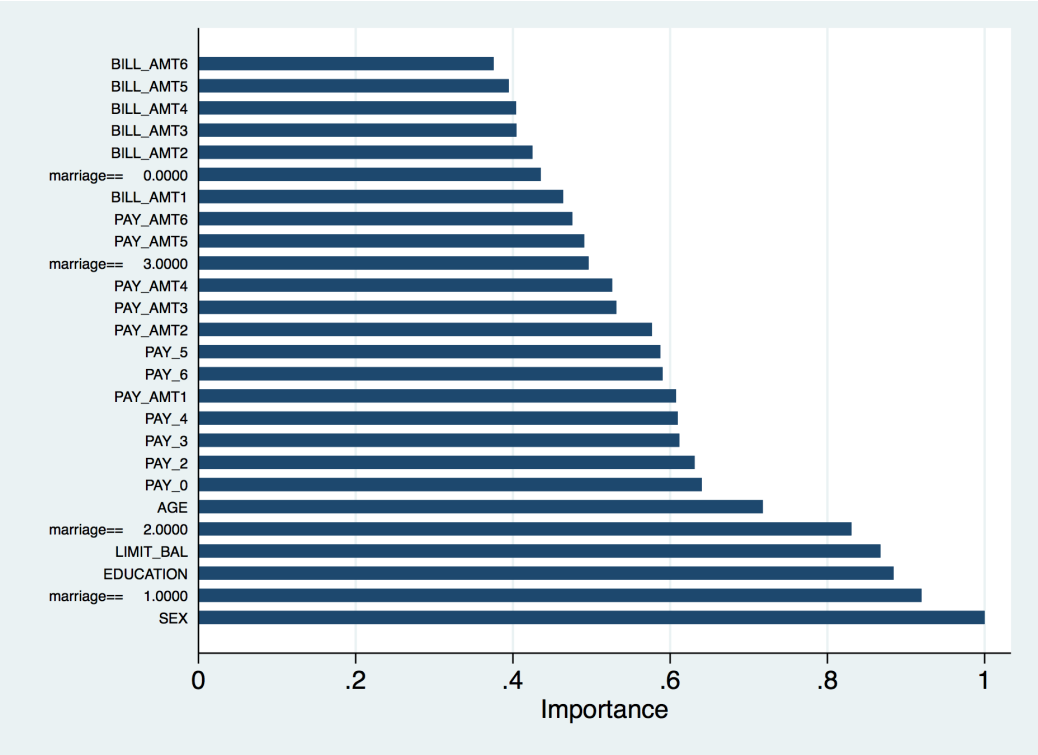

Figure 5: Importance Score of Predictor Variables

We can see from the plot that the top 5 most important predictors are basic demographic and background information such as gender, education, and marital statuses corresponding to "married" (marriage== 1.000) and "single" (marriage== 2.000). We can also see that none of the variables encoding monthly bill amounts (bill_amt) is particularly important, comparing with the rest of the predictors. Surprisingly, however, the amount of monthly spending limit (limit_bal) is the fourth most important predictor in the random forest model. We can overlay two histograms of the monthly spending limit to obtain more insights on how this variable affects the response variable:

```
twoway (hist limit_bal if defaultpaymentnextmonth == 0) ///
(hist limit_bal if defaultpaymentnextmonth == 1, ///
fcolor(none) lcolor(black)), legend(order(1 "no default" 2 "default" ))
```

We can see from the histograms that card holders who default on their debt generally have a lower monthly spending limit than those who do not default.

5.3 Comparison with Logistic Regression

Alternatively, credit default can be modeled using logistic regression. The following code returns the prediction accuracy of logistic regression, using the same set of predictor variables and the same train/test split:

```
logistic defaultpaymentnextmonth limit_bal sex education \\\
```

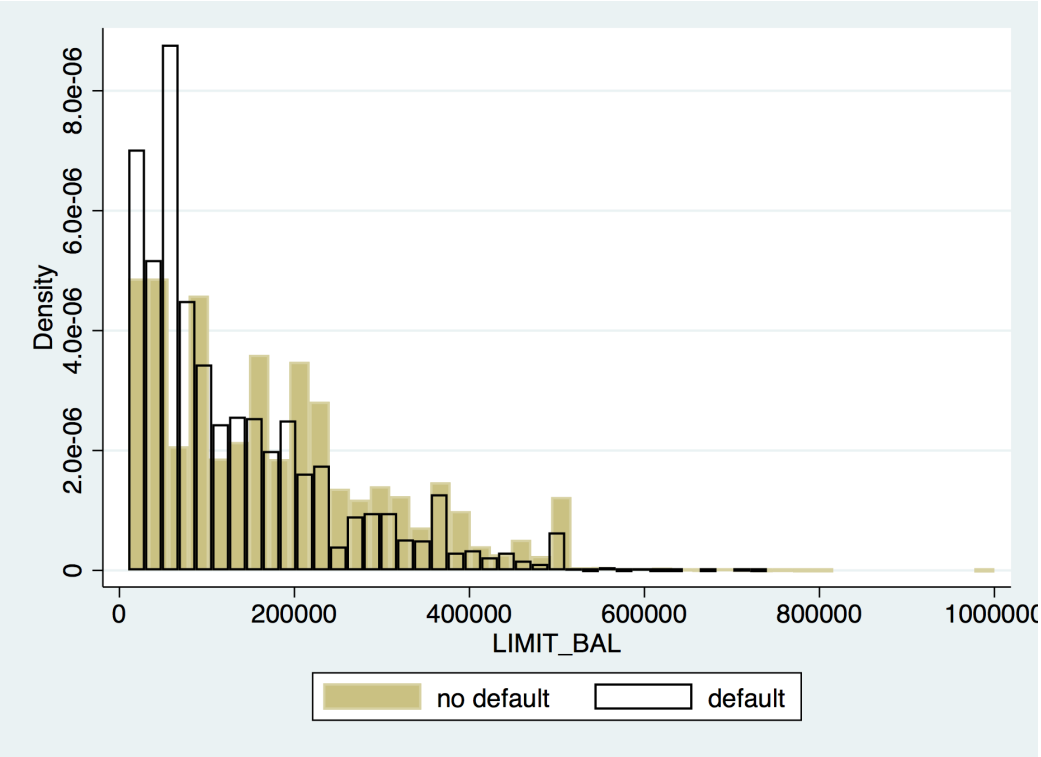

Figure 6: Histograms of Monthly Spending Limit

```
marriage_enum* age pay* bill* in 1/15000

predict plogit in 15001/30000
replace plogit = 0 if plogit <= 0.5 & plogit != .
replace plogit = 1 if plogit > 0.5 & plogit != .
gen error = plogit != defaultpaymentnextmonth
sum error in 15001/30000
```

The prediction error obtained using logistic regression is 18.72%, comparing with the best-so-far error rate that we have from random forest, which is 18.01%. This translates to a total of $(18.72\% - 18.01\%) \times 15000 \approx 106$ incorrectly classified observations. In a real-life situation, the 0.71% difference in error rate could lead to millions of dollars saved due to improved credit card default prevention.

6 Example: Online News Popularity

The following sample data comes from a 2015 presentation at the Portuguese Conference on Artificial Intelligence (Fernandes, Vinagre, & Cortez, 2015; Dheeru & Karra Taniskidou, 2017).[3] There are a total of 39,644 observations, 1 response variable, and 58 explanatory variables. For this problem, we are interested in the log-scaled number of "shares" an online article obtains based on various nominal and continuous attributes such as whether the article was published on a weekend, whether certain keywords are present, number of images in the article, and etc. For a full list of variable names and descriptions, please refer to Appendix B.

[3]To access the exact data set used in this example, please visit
`https://archive.ics.uci.edu/ml/datasets/Online+News+Popularity`

6.1 Model Training and Parameter Tuning

First we need to randomize the data like we did for the previous classification example. Then generate a new variable for log-scaled number of shares:

```
use OnlineNewsPopularity
set seed 201807
gen u = uniform()
sort u
gen logShares = ln(shares)
replace logShares = 0.01 if logShares == .
```

We will use a 50-50 split to partition the data into training and testing set as in the previous example. To tune the hyper parameters *numvars* and *iterations*, we employ the same technique as in the previous example where we fix the value of one hyper parameter when tuning the other. This is a viable parameter optimization method due to the fact that error rate for random forest converges when the number of iterations is large enough. Essentially, our goal is to set a reasonably large number of iterations where the out-of-bag and validation errors converge so that when we tune the number of randomly selected variables, we can ascertain that the errors differ due to the value of `numvars` and not due to `iterations`. We will again start with `iterations` = 10 and increase it by increments of 5 until `iterations = 100`. At the end of the loop, we plot the out-of-bag errors and the actual RMSE values validated using the test data against the number of iterations.

```
gen out_of_bag_error1 = .
gen validation_error = .
gen iter1 = .
local j = 0
forvalues i = 10(5)100 {
    local j = `j' + 1
    randomforest logShares n_* average_* num_* ///
    data_* kw_* self_* weekday_* lda_* global_* ///
    is_weekend rate_* min_* max_* avg_* title_* abs_* in 1/19822, ///
    type(reg) iter(`i') numvars(1)
    replace iter1 = `i' in `j'
    replace out_of_bag_error1 = `e(OOB_Error)' in `j'
    predict p in 19823/39644
    replace validation_error = `e(RMSE)' in `j'
    drop p
}
label var out_of_bag_error1 "Out of Bag Error"
label var iter1 "Iterations"
label var validation_error "Validation RMSE"
scatter out_of_bag_error1 iter1, mcolor(blue) msize(tiny) || ///
scatter validation_error iter1, mcolor(red) msize(tiny)
```

We can see from the graph that the OOB error and validation RMSE start to converge to a fixed value at 80 iterations. We get the lowest value for both errors at 100 iterations, which will be set for the final model. Now we can tune the other hyper parameter, `numvars`, to see which one gives the lowest validation RMSE.

```
gen oob_error = .
gen nvars = .
gen val_error = .
```

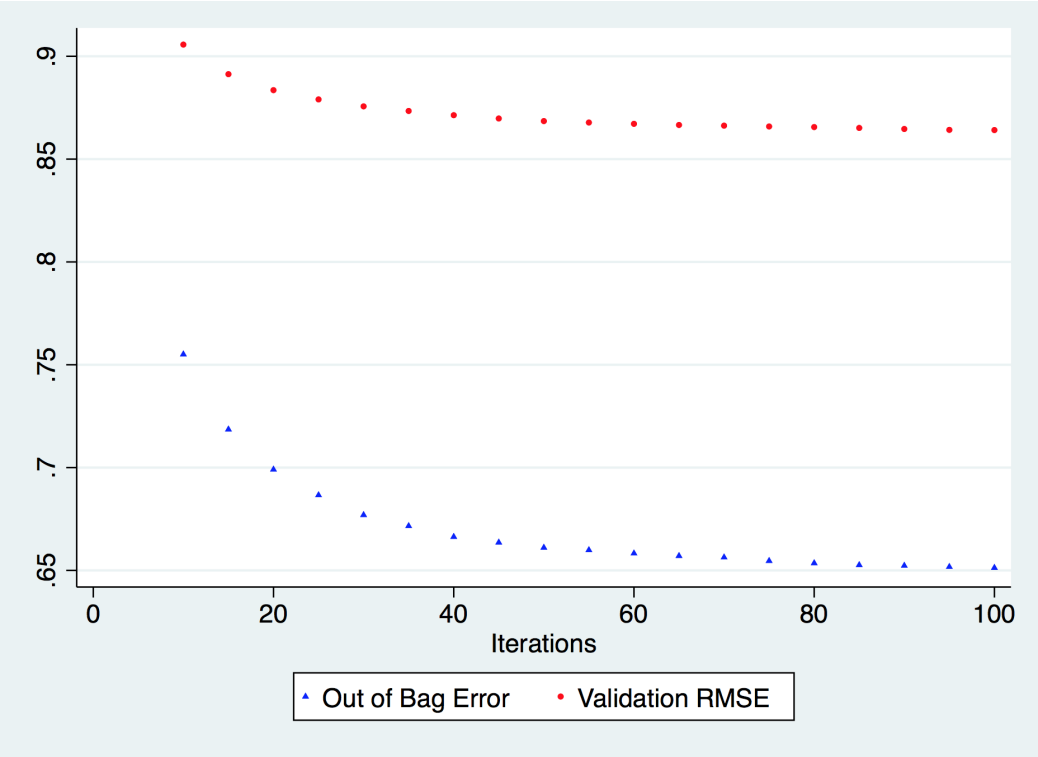

Figure 7: Out of Bag Error and Validation RMSE vs. Iterations Plot

```
local j = 0
forvalues i = 1(1)58{
    local j = `j' + 1
    randomforest logShares n_* average_* num_* ///
    data_* kw_* self_* weekday_* lda_* global_* ///
    is_weekend rate_* min_* max_* avg_* title_* abs_* in 1/19822, ///
    type(reg) iter(100) numvars(`i')
    replace nvars = `i' in `j'
    replace oob_error = `e(OOB_Error)' in `j'
    predict p in 19823/39644
    replace val_error = `e(RMSE)' in `j'
    drop p
}
label var oob_error "Out of Bag Error"
label var val_error "Validation RMSE"
label var nvars "Number of Variables Randomly Selected at Each Split"
scatter oob_error nvars, mcolor(blue) msize(tiny) || ///
scatter val_error nvars, mcolor(red) msize(tiny)
```

Using the commands `sum val_error` and `list val_error`, we can see that the lowest validation error is RMSE = 0.856689, obtained at `numvars = 5`, which will be set for our final model.

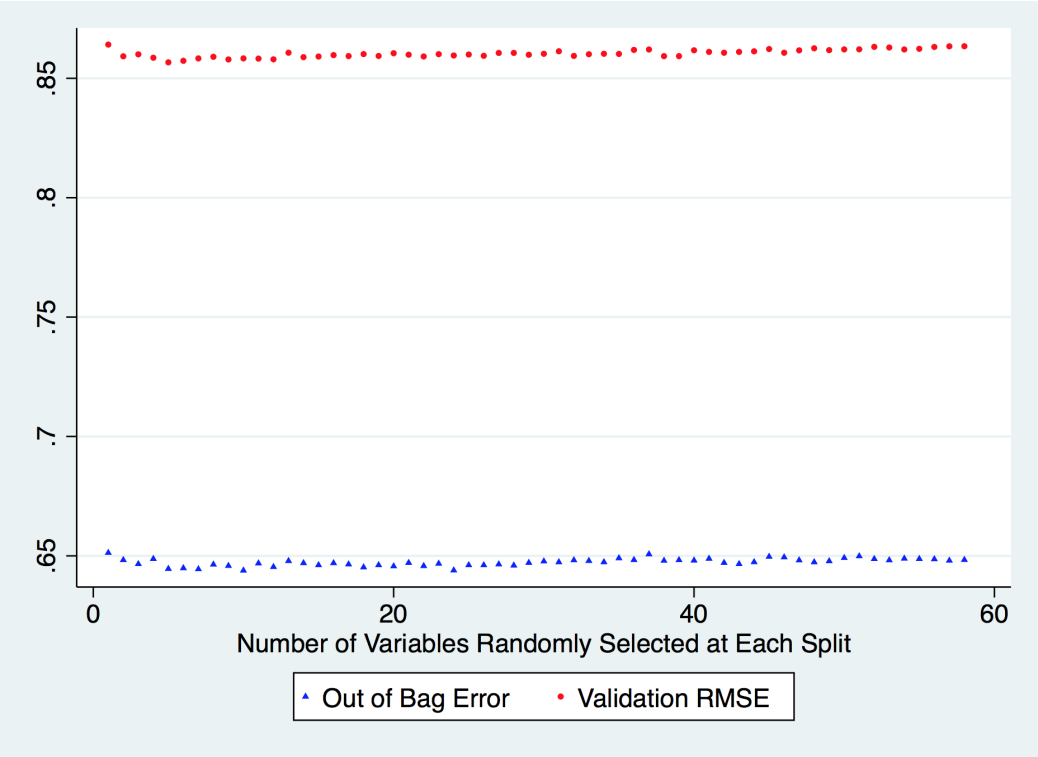

Figure 8: Out of Bag Error and Validation Error vs. Number of Variables Plot

6.2 Final Model and Interpretation of Results

The final model has hyper parameter values $numvars = 5$ and $iterations = 100$.

```
randomforest logShares n_* average_* num_* ///
data_* kw_* self_* weekday_* lda_* global_* ///
is_weekend rate_* min_* max_* avg_* title_* abs_* in 1/19822, ///
type(reg) iter(100) numvars(5)
eret li

predict prf in 19823/39644
eret li
```

The final out-of-bag error is 0.644592. This is slightly larger than the RMSE calculated against the testing data, which is 0.856689. To learn which variables affect the prediction accuracy, we can generate a variable importance plot using the same code segment as the previous classification example. In the following variable importance plot, only the top 20 important predictors are shown in order to make the labels more legible. We can see that the categorical variable encoding whether or not the article was published on a weekend is the most important predictor. Other important explanatory variables include news channel types and the number of keywords. To obtain more insight on how the log-scaled number of article shares is related to whether the article was published on a weekend, we use the following histogram to illustrate the relationship:

```
twoway (hist logShares if is_weekend == 0) ///
(hist logShares if is_weekend == 1, fcolor(none) lcolor(black)), ///
legend(order(1 "weekday" 2 "weekend" ))
```

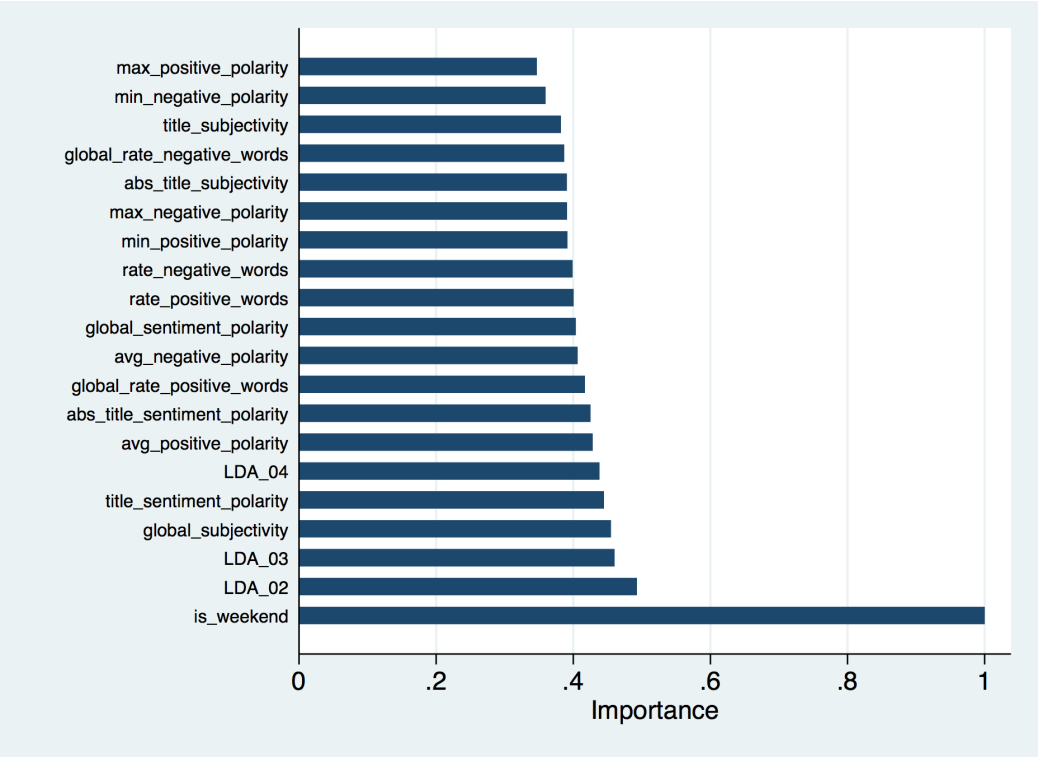

Figure 9: Variable Importance Plot

We can see how the empirical distributions of log number of shares differ for weekdays vs. weekends. This clear shift in empirical distribution helps to explain why the `is_weekend` explanatory variable was the most important in the model.

6.3 Comparison with Linear Regression

The following code block fits a linear regression model over the same set of dependent and independent variables, using the same train/test split as shown in the random forest model:

```
regress logShares n_* average_* num_* ///
data_* kw_* self_* weekday_* lda_* global_* ///
is_weekend rate_* min_* max_* avg_* title_* abs_* in 1/19822

predict pregress in 19823/39644
ereturn list
```

The value of `e(rmse)` displayed is the RMSE calculated over the training data. To compare the linear model with random forest, we need to calculate the RMSE over the testing data using the following commands:

```
gen diff_sqr= (logShares - pregress)^2
summarize diff_sqr
```

We can see from the output that the mean squared error is 40.90379, which means the RMSE is equal to $\sqrt{40.90379} \approx 6.3956$, which is much higher than the RMSE fitted over the training data.

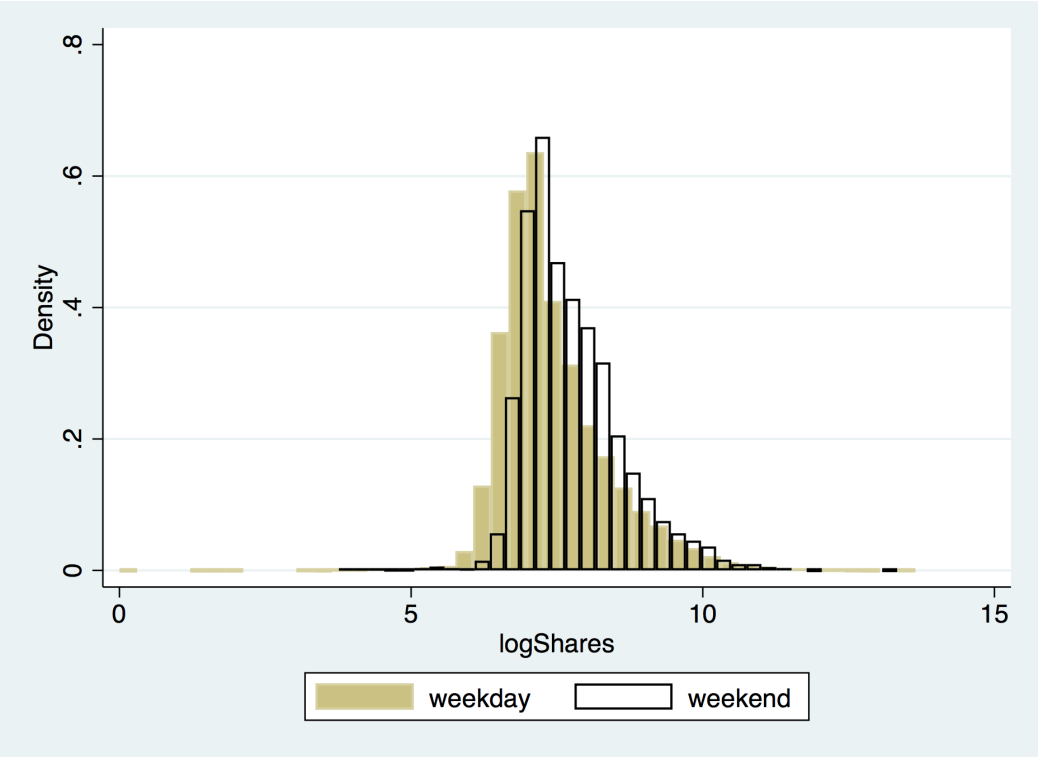

Figure 10: Histograms of Log-scaled Number of Shares

Comparing with the testing RMSE obtained from the random forest model, the testing RMSE for the linear model is much higher. This is a strong indication that for this example, random forest out-performs linear regression.

7 Discussion

The classification and regression examples have illustrated that random forest models usually have higher prediction accuracy than corresponding parametric models such as logistic regression and linear regression. Typically, greater gains in model performance are available for multi-class (multinomial) outcomes and regression than binary outcomes. For example, Ing et al. (2018) found that support vector machines did not improve over logistic regression. This is also the case for random forest, as we have demonstrated in the classification example.

In the examples, the values of hyper parameters were determined based on which value gave the lowest testing error. In practice, when there are not enough observations to allow for a train/test split, the OOB error can be used instead. As previously demonstrated, the OOB error is a close estimation of the actual testing error and can be used on its own as a criterion for parameter tuning.

While the two examples primarily focused on the typical case of tuning `iterations` and `numvars`, depending on the data set and software constraints, other hyper parameters such as max tree depth and minimum size of leaf nodes could be taken into consideration during parameter tuning. For instance, setting the max tree depth to a fixed value may become necessary on a machine with limited RAM.

8 Acknowledgment

The software development in Stata was built on top of the WEKA Java implementation, developed by the University of Waikato. We are grateful for Eibe Frank for allowing us to use the WEKA implementation for the Stata plugin.

This research was supported by the Social Sciences and Humanities Research Council of Canada (SSHRC # 435-2013-0128).

References

Basuchoudhary, A., Bang, J. T., & Sen, T. (2017). *Machine-learning Techniques in Economics: New Tools for Predicting Economic Growth*. New York: Springer International Publishing.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Dheeru, D., & Karra Taniskidou, E. (2017). *UCI Machine Learning Repository.* Retrieved from `http://archive.ics.uci.edu/ml`

Fernandes, K., Vinagre, P., & Cortez, P. (2015, May). A proactive intelligent decision support system for predicting the popularity of online news. In F. Pereira, P. Machado, E. Costa, & A. Cardoso (Eds.), *Proceedings of the 17th Portuguese Conference on Artificial Intelligence* (pp. 535–546). New York: Springer.

Frank, E., Hall, M., & Witten, I. (2016). The WEKA Workbench Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques". *Morgan Kaufmann*.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, *11*(1), 10–18.

Ho, T. K. (1995, August). Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition* (Vol. 1, pp. 278–282). Piscataway, NJ: IEEE. doi: 10.1109/ICDAR.1995.598929

Liu, X., Wu, D., Zewdie, G. K., Wijerante, L., Timms, C. I., Riley, A., ... Lary, D. J. (2017). Using machine learning to estimate atmospheric Ambrosia pollen concentrations in Tulsa, OK. *Environmental Health Insights*, *11*. doi: 10.1177/1178630217699399

Nyman, R., & Ormerod, P. (2017). Predicting Economic Recessions Using Machine Learning Algorithms. *arXiv preprint arXiv:1701.01428*.

Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, *5*(1), 3–55.

Yeh, I.-C., & Lien, C.-h. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, *36*(2), 2473–2480.

Appendix A Variable Names for Classification Example

The column names in this table are reproduced based on the original documentation on UCI Machine Learning Repository's website.

Variable Name	**Column Name**
id	row number
limit_bal	Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
sex	Gender of the card holder. 1 = male, 2 = female
education	Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
marriage	Marital status (1 = married; 2 = single; 3 = others).
age	Age of card holder
pay_0	The repayment status in September, 2005 The value of the variable corresponds to number of months delayed.
pay_2	The repayment status in August, 2005
pay_3	The repayment status in July, 2005
pay_4	The repayment status in June, 2005
pay_5	The repayment status in May, 2005
pay_6	The repayment status in April, 2005
bill_amt1	Amount of bill statement in September, 2005
bill_amt2	Amount of bill statement in August, 2005
bill_amt3	Amount of bill statement in July, 2005
bill_amt4	Amount of bill statement in June, 2005
bill_amt5	Amount of bill statement in May, 2005
bill_amt6	Amount of bill statement in April, 2005
pay_amt1	Amount of previous payment in September, 2005
pay_amt2	Amount of previous payment in August, 2005
pay_amt3	Amount of previous payment in July, 2005
pay_amt4	Amount of previous payment in June, 2005
pay_amt5	Amount of previous payment in May, 2005
pay_amt6	Amount of previous payment in April, 2005
defaultpaymentnextmonth	Whether the card holder defaults on his/her payment next month; 0 = no, 1 = yes
marriage_enum1	Marital status is none of "married", "single", or "other"; generated during data pre-processing
marriage_enum2	Marital status = "married"; generated during data pre-processing
marriage_enum3	Marital status = "single"; generated during data pre-processing
marriage_enum4	Marital status = "other"; generated during data pre-processing

Appendix B Variable Names for Regression Example

The column names in this table are reproduced based on the original documentation on UCI Machine Learning Repository's website.

Variable Names	**Column Names**
url	URL of the article (non-predictive)
timedelta	Days between the article publication and the dataset acquisition (non-predictive)
n_tokens_title	Number of words in the title
n_tokens_content	Number of words in the content
n_unique_tokens	Rate of unique words in the content
n_non_stop_words	Rate of non-stop words in the content
n_non_stop_unique_tokens	Rate of unique non-stop words in the content
num_hrefs	Number of links
num_self_hrefs	Number of links to other articles published by Mashable
num_imgs	Number of images
num_videos	Number of videos
average_token_length	Average length of the words in the content
num_keywords	Number of keywords in the metadata
data_channel_is_lifestyle	Is data channel 'Lifestyle'?
data_channel_is_entertainment	Is data channel 'Entertainment'?
data_channel_is_bus	Is data channel 'Business'?
data_channel_is_socmed	Is data channel 'Social Media'?
data_channel_is_tech	Is data channel 'Tech'?
data_channel_is_world	Is data channel 'World'?
kw_min_min	Worst keyword (min. shares)
kw_max_min	Worst keyword (max. shares)
kw_avg_min	Worst keyword (avg. shares)
kw_min_max	Best keyword (min. shares)
kw_max_max	Best keyword (max. shares)
kw_avg_max	Best keyword (avg. shares)
kw_min_avg	Avg. keyword (min. shares)
kw_max_avg	Avg. keyword (max. shares)
kw_avg_avg	Avg. keyword (avg. shares)
self_reference_min_shares	Min. shares of referenced articles in Mashable
self_reference_max_shares	Max. shares of referenced articles in Mashable
self_reference_avg_sharess	Avg. shares of referenced articles in Mashable
weekday_is_monday	Was the article published on a Monday?
weekday_is_tuesday	Was the article published on a Tuesday?
weekday_is_wednesday	Was the article published on a Wednesday?
weekday_is_thursday	Was the article published on a Thursday?
weekday_is_friday	Was the article published on a Friday?
weekday_is_saturday	Was the article published on a Saturday?
weekday_is_sunday	Was the article published on a Sunday?
is_weekend	Was the article published on the weekend?

Continued from last page

Variable Names	**Column Names**
LDA_00	Closeness to LDA topic 0
LDA_01	Closeness to LDA topic 1
LDA_02	Closeness to LDA topic 2
LDA_03	Closeness to LDA topic 3
LDA_04	Closeness to LDA topic 4
global_subjectivity	Text subjectivity
global_sentiment_polarity	Text sentiment polarity
global_rate_positive_words	Rate of positive words in the content
global_rate_negative_words	Rate of negative words in the content
rate_positive_words	Rate of positive words among non-neutral tokens
rate_negative_words	Rate of negative words among non-neutral tokens
avg_positive_polarity	Avg. polarity of positive words
min_positive_polarity	Min. polarity of positive words
max_positive_polarity	Max. polarity of positive words
avg_negative_polarity	Avg. polarity of negative words
min_negative_polarity	Min. polarity of negative words
max_negative_polarity	Max. polarity of negative words
title_subjectivity	Title subjectivity
title_sentiment_polarity	Title polarity
abs_title_subjectivity	Absolute subjectivity level
abs_title_sentiment_polarity	Absolute polarity level
shares	Number of shares (target)