FINAL PROJECT (MORE THAN A CORGI CLUB)

Jin Kweon and Clover Jeong

12/09/2017

Group members: Jin Kweon and Jiyoon Clover Jeong

Goal

Based on given 15 variables on individual samples, perform prediction to determine whether a person makes over 50K a year.

Abstract & Motivation & Introduction & Outline

The motivation is to think of and predict what factors set the income for each person from the 1994 Census database, based on categorical and continuous variables: age, workclass, fnlwgt, education, education-num, marital-status, occupation, relationship, race, sex, captial-gain, capital-loss, hours-per-week, native-country, and income.

Throughout the work of the project, our group (More than a corgi club) tries to find out the how the given 15 variables affect people's incomes. Our group is formed of two students: Jin and Clover. We have used R language, to come up with the answers. All the files and results are uploaded to Box folder (and our group's Github), with the link following: Box for More than a corgi club and Github for More than a corgi club. The types of file extensions are ".R," ".Rmd," ".md," ".html," ".png," ".csv," ".txt," and ".pdf."

Our group performed classfication decision tree, random forest, bagged tree, random forest, boosted tree, to achieve our goal.

Also, plese be aware appendix does not count for 20 page limits.

Problems & Research questions

Our analysis are focusing on the following questions:

- Which variable does have to do with the income the most?
- Which model is the best in terms of having less testing error rate?
- Which model is the best in terms of having higher AUC?

From these analytical / exploratory research questions above, our group came up with an answer for the main question/goal of our project: "How can we make (What is the) the best prediction model to determine a person makes over 50K a year?"

So, basically our goal is to make the prediction error rate reasonably good enough and make the model not too much complicated (so we can actually interpret the models well).

Data

Our group went over the variable names whether we could understand all, and we had found the three columns: "capital-gain," "capital-loss," and "fnlwgt" were unclear to us. Fortunately, we could find the definition of three variables. "fnlwgt" variable standed for final weight, and it had to do with demographic characteristics. Also, "capital-gain" and "capital-loss" meant "income from investment sources, apart from wages/salary" and "losses from investment sources, apart from wages/salary," respectively.

The dimensions of training and testing sets were 32561 by 15 and 16281 by 15, respectively. As you could see, the training and testing sets were already given to us, so we did not need to bother separating big data sets randomly into training and testing again (but we might need to randomly divide training into another training and testing sets later). There were some missing values in both training and testing sets, and our group would talk about in details how we dealt with them in EDA part.

Here were good explanations of 15 variables we had, below:

age: age (continuous)

workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked (categorical) - 8 levels

fnlwgt: final weight (continuous)

education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool (categorical - the highest education) - 16 levels

education-num: education year (continuous)

marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse (categorical) - 7 levels

occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces (categorical) - 14 levels

relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried (categorical) - 6 levels

race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black (categorical) - 5 levels

sex: Female, Male (categorical) - 2 levels

capital-gain: income from investment sources, apart from wages/salary (continuous)

capital-loss: losses from investment sources, apart from wages/salary (continuous)

hours-per-week: work hours per week (continuous)

native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands (categorical) - 40 levels

income: >50K, <=50K (categorical) - 2 levels

Please refer to the appendix for more details about data.

Preprocessing and Exploratory Data Analysis

This EDA was performed before any analysis, to learn about the data. Our team spent a lot of time on this part. We could clean both training and testing sets (literally, it was just pre-processing data), but we could not really do any machine learning techniques on the testing set, since it will be biased/cheating.

1. Find missing value/NA (Should be performed on both training and testing sets)

The data had some missing values.

Here were summaries where we could find about missing values:

For the training set, there were 1836 missing values in "workclass" variable, 1843 missing values in "occupation" variable, and 583 missing values in "native-country" variable. For the testing set, there were 963 missing values in "workclass" variable, 966 missing values in "occupation" variable, and 274 missing values in "native-country" variable. There were different ways to deal with missing values: delete the rows, randomly assign the values, plug in the mean/median/mode, use the predict function to predict values, etc.

Also, our group had computed the percentage of missing values for each column, and we found that "workclass" and "occupation" variables were missing around 6% and "native-country" variable was missing around 1.5% (for both training and testing). Although it was not significant number of missing values, we decided not to delete the rows, because if the missing values for each column were never overlapped each other, then, we might end up deleting the rows of around 12% of the data, and lost a lot of prediction power in the future.

We had found so many different packages to deal with missing values, and here were some that we found the most interesting and the most helpful to our data sets: mice, missForest, mlr, Hmisc, and mi (by the way, we found the interesting fact that random forest in R used mean/mode imputation for missing values).

From the histograms we plotted, in the training set, there were 30162 observations with no missing value, 7 observations with "occupation" variable missing only, 556 observations with "native-country" variable mising only, 1809 observations "workclass" and "occupation" variables missing together, 27 observations all three of them missing (so, we at least knew there was no observation "workclass" variable missing only from the training set). - total 2399 observations containing at least one missing value

We examined missing values even further. In the testing set, there were 15060 observations with no missing value, 3 observations with "occupation" variable missing only, 255 observations with "native-country" variable missing only, 944 observations "workclass" and "occupation" variables missing together, 19 observations all three of them missing (so, we at least knew there was no observation "workclass" variable missing only from the testing set). - total 1221 observations containing at least one missing value

Most of the packages we have tried just took too much time as our data was large. And, we believed non-parametric missing value imputation was better in our data set, we decided to use missForest package to retrieve missing values. After retrieving with missForest, we found that categorical variables were imptued with around 12.5% error (not small, but still better than assigning mode - "assigning mode is terrible idea, as this was just assigning one value into every mistery missing element, and it could cause as much as problems as just deleting rows that contain missing elements" or other methods) for both training and testing sets, and we could improve it by tuning ntree and mtry parameters.

Since whenever we re-run/knit the rmd file, it would take us so much time to run missForest function (around 30 minutes), so we rather saved new data in csv file in "/data/cleandata" folder after imputing (and made missForest function as comment).

Last but not least, we checked whether the missForest worked fine by randomly checked the retrieved rows. And, from now on, we were going to use "newtrain" and "newtest" for any analysis.

2. Take out the variables that are not interpretable

We were not going to take out any variable, as we researched and understood the definitions of each variable.

3. Find out continuous variables, and transform them into numeric if necessary

No further changes were needed.

4. Find out categorical variables, and transform them into factor if necessary

No further changes were needed.

5. Handling outliers/weird data

There was a way to handle outliers for categorical variables, but our group would not decide to work on that, as we did not know how the data was collected. It made more sense for us to fix the outliers if the person who collected the data was supposed to the similar distribution for every level of each categorical variable; however, we did not know it. So, we assumed that it made sense to have different density distributions for each level.

And, even for continuous variables, we did not know whether the so called "outlying datas" were bad (wrongly inputed) or not. So, our logical thoughts were really important, and we relied on it, here. We looked at boxplots and density plots to help us make decisions whether we needed to remove outlying observations or not.

First of all, we wanted to say that we printed out the lists of outlying datas we got from usual mathematical way (we usually say outliers were the observatoins above or below Q1/Q3 $\pm$ IQR * 1.5); however again, since we did not know the distribution assumption, we were going to approach this problem with our logics.

Training set

So, we decided that there was no outlier for "age" variable (the distribution of the "age" variable was reasonable). And, we decided that there was no outlier for "education num" variable (it might be easy for us to just say there are four outliers: 2885th, 2947th, 3447th, and 3593th observations that had 1-year-long education year, but when we checked their backgrounds and occupations, we concluded they were not actually outliers). Also, we decided that there was no outlier for "hours per week"

variable (the data pretty looked like mean-centered and this logically made sense). And, we concluded that there was no outlier for "capital loss" variable. Furthermore, we did not see any outlying datas for "fnlwgt" variable.

However, we found "capital gain" variables looked suspicous to us. When we looked at the tables of capital gain, there were 159 people earning 99999 US dollars, and it was obviously did not make sense (these data might indicate that they were the ones who gained above certain amount of capital gain, but stil did not make sense to have 99999 dollars for 159 people. Also, when we dealt with the data, people usually inputed "-1," "999," or something not make sense. And, our group believed they were definitely weird data). The second group was earning 41310 US dollars (also, compared with capital loss data, 99999 US dollars gain did not make sense at all). So, we rather decided to remove 159 observations as we thought that the data was mistakenly collected.

So people might ask us why not just retrieve these 159 observations. Here were the reasons why we removed them. First, it was small portion of our data. They were not even closed to 1% of our data, and these observations would not have significant impacts (compared to this, the number of missing values were pretty a lot). Second, it could be dangerous to retrieve the data. Although we retrieved the data with any good algorithms, they could never be the true data, and we were afraid that retrieving too often could cause even small bias. Also, there was always a possibility that retreiving algorithm could create another unexpected outliers. Third, there was no guarantee for retrieving the data would help our analysis. Last but not least, we were little bit suspicious of not only these one "capital gain" entry was wrong, but it might be possible that some of the other entries were wrongly inputed. By looking at these 159 observations, we realized that it was possible for the person who collected data miss some other entries' information as well. Since 99999 dollars of "capital gain" definitely did not make sense and it could be easily spotted by readers, the data collector might just input 99999 dollars for "capital gain" for giving us hints that many other entries were wrong for these observations (for example, if he/she missed "age" variable, he/she could just got mean/median to retrieve "age" data, and inputed 99999 dollars for "capital gain").

So, now, the dimension of newtrain data set was 32402 by 15. Now, our plots looked more sense.

Testing set - since we belived the data was wrongly collected, we were going to clean the testing set as well.

Similar with the training sets (it should be, as training and testing sets were randomly splitted into two), the testing set also showed outlying data in "capital gain" variable. This time, they had 85 observations that earned 99999 US dollar, and as we did before, we removed 85 unexpecting observations.

Another thing we were really amazed when we plotted the boxplots and density plots, was the plots all really looked the same between training and testing sets (it might imply that if we trained **well** on the training set, we hoped to see great prediction accuracy on the testing set). We could confirm that they were well randomly splitted.

So, now, the dimension of newtest data set was 16196 by 15. Now, our plots looked more sense.

Also, in conclusion, we thought there might be better ideas than simply removing 159 and 85 suspicious data from training and testing sets, but since we did not have any much information of prior distributions and they were just small portions of training and testing sets (0.488% and 0.522% respectively), we rather simply removed them.

6. Do str and summary to find out data structures and fix if necessary

When we read the structures and summaries for both training and testing sets, nothing looked abnormal, so we did not make any transformation (or modifying data in other way).

- Here were some summaries of data sets:

Most frequent elements of the workclasses were "Private," the gap between max and min for "fnlwgt" was pretty huge (but it might be possible, since this number indicated demographic characteristic), the most frequent element of the "workclass" was "HS-grad" (this might not make sense if this was collected nowadays, but remember that this data set was collected in 1994 and donated in 1996, and at that time, not many people went to colleges), the most frequent element of the "race" was "white," the number of male was twice as much as female, and last but not least, around 76% of the collected observations earned less than or equal to 50K.

7. Discretizeing/Binning (Should be performed on both training and testing sets)

Although one of the advantages of decision tree was that data did not have to be pre-processed; however, discretizing could help us interpret features in the future.

Click here: Fig 25 ~ 28 to see the plots in appendix.

Categorical

First of all, we needed to think whether all **categorical variables** should be dummified or it could be better to do it with discretizied. We could usually consider the categorical variable to be discretized when there were enough number of levels and the data were skewed (so, the variable "sex" was better to be dummified than being discretized, since there were only two levels, and there was no point for us to discretize).

When we saw the "workclass" variable (for both training and testing), there were really small number of "Never-worked" and "Without-pay" levels/categories, so we decided to combine them together (and this made sense for interpretation as well, since "Never-worked" is kind of similar to "Without-pay") as "No-gain" category.

For "native country" variable, around 91% (for both training and testing sets) of the observations were United-States, meaning that all other 40 countries/levels had relatively small impact/numbers in the variable (also, there were many Mexico and Philippines - 2% and 1% respectively). So, we decided to discretize United States v.s. Mexico v.s. Philippines v.s. all other countries.

Also, we decided to leave "marital.status," "occupation," "relationship," "race," and "sex" as the way they were (because some of them were well-spreaded, we might find some were unnecessary go through this process, or we might believe leaving them help us interpret in the future).

Continuous

For continuous variables, we were **not** going to discretize any of them for **interpretation purpose** (also many of variables were skewed/concentrated, so we believed it would be better not to discretize).

8. Convert to dummy indicators (Should be performed on both training and testing sets)

From pg 373-377 in *APM*, it says *"For classification trees using CART, there is no practical difference in predictive performance when using grouped categories or independent categories predictors."* However, we decided to go for independent categories/1-hot encoding because of the two following reasons:

1. Independent categories method is easier to interpret.
2. As pg 377 of APM says, independent category predictors provide valuable interpretation about relationship between predictors and response.

So, we dummified "workclass," "marital status," "occupation," "relationship," "race," and "sex," and "native country" (7 variables in total).

Binning: "workclass" and "native country"

Dummify: "workclass," "marital status," "occupation," "relationship," "race," "sex," and "native country"

At this time, we decided to make another new cleaned data sets: newtrain2 and newtest2.

9. Extra EDAs we did...

We found that the variables "education" and "education.num" exactly meant the same thing, meaning that they could cause the collinearity issues. So, we decided to take out "education" variable, since we preferred to leave continuous variable, which we believed easier to interpret (The number of levels were the same, so we started to be suspicous. And, we could conclude collinearity issue when we looked at the frequency proportions for each level). Click here: Fig 29 ~ 30 to see the tables in appendix. Also, we got a correlation matrix for continuous variables, to check any collinearity issue. And, we concluded that there was no more collinearity issue after we removed "education" variable.

Also, we made plots between each predictor and "income" (response) variable, to see the association between each predictor and response variable.

Also, we found that there are dot (.) for income of the testing sets only, so we decided to remove them.

Also, we decided not to scale any variable at this momeent, because we wanted to keep the original unit scale, and also, it was easier to interpret in the future. We also explained specifically why we did not need to scale for some tree methods later in the "Bagged Tree" section.

Furthermore, we briefly introduced the definition of "fnlwgt" (final weight) variable under the data section of our report above. Based on the definition, this weight variable would not be a good predictor to include in our analysis (but this weight variable would might help us during the analysis). So, for now, we would remove this variable.

Last but not least, we might remove "native.country" varible in the future, as we were little bit suspicous whether this variable was actually contributing anything for classification (meaning that it might not be a good predictor).

Methodology & Analysis

Important things to know before Methodology & Analysis!!!

1. We fixed the set.seed as 100 for every method, so we could always reproduce our outputs.
2. In the "images" folder, some of the numbers are skipped because we assigned the number on some diagram, but decided not to use it. For example, let's say we included the pictures named "report34," "report35," "report36," etc. However, we decided to remove "report35" and "report36," then, we simply removed them and we did not rearrange other following plots (for example, we did not change "report37" and "report38" to "report35" and "report36"). Hope this does not confuse you.
3. We decided to use both AUC (or we can say ROC) and accuracy rate for the model selection. Definitely, our goal of this project was to improve prediction accuracy, so it made sense for us to go with accuracy rate. However, when we looked at our entire data (the one combined both training and testing), our response variable was imbalanced. Around 76.5% of our data was earning less than 50K and the rest 23.5% is earning more than 50K. As we learned in the class, as there was any data imbalance, it was recommended to use AUC (or ROC) for model selection. So, our answer was to use **both of them** and select one model with the highest AUC and one with the highest accuracy. The best scenario was when these two were the same. For accuracy rate, we were going to get the optimal threshold (the threshold with the smallest error rate) only with the train set. Around 0.5 usually gave the smallest misclassification rate (that was what we found, as you could find later). Also we thought that two different types of errors - false positive and false negative errors were equally bad, so 0.5 for the threshold made more sense here. And then, we predicted the accuracy rate with the testing set.
4. We decided not to show some of the methods such as random forest and boosted trees, since they were "literally" forests (we could if we wante to).
5. Due to the time limit, we might need to take "greedy approach," and instead of comparing all the parameters, we **might** need to fix one parameter and train the other parameters.
6. To get the optimal threshold to predict in the testing set, instead of doing cross validation (because it took so much time), we plotted misclassification error rate in the training set v.s. thresholds, and got the the threshold where misclassification was the lowest.
7. Due to the page limits, we minimized our plots, and it might be hard for you to see it. So, please use pdf and zoom it up for reading.

Here were comprehensive lists of tuning parameters for each methods.

1. Classification tree: alpha for cost complexity pruning, classification for splitting tree
2. Bagged tree: number of trees (using tuning parameter alpha for cost complexity), depth of the tree (~minimum size of nodes)
3. Random forest: number of predictors considered, number of trees, depth of the tree (~minimum size of nodes)
4. Boosted tree: number of splits/interaction depth, shrinkage parameter, number of trees

There was one more thing we wanted to pointed out before we talked about the analysis of Tree methods. Although there were sets of number of tuning parameters, it was possible for us improve the test prediction error by tuning other parameters as well. For example, we usually tuned only alpha from the cost-complexity pruning for classification tree, but we could actually tune something else such as minimum number of observations in terminal node.

And, we specifically wrote out which tunining parameters we used for each method.

Some of our methods took a lot of time for tuning parameters, and we needed to think between how much we needed to tune the parameters and how much we needed to simplify the model. Since our goal of this project was to make the prediction error rate reasonably good and make the model simple enough for interpretation, we performed reasonable amounts of tunings.

1. Build a Classification Tree

Classification tree is one of the most famous non-parametric classification methods. We are performing a classification tree instead of regression tree, we try to predict a qualitative response: income (two categories).

Strengths:

- Decision trees can be constructed in the presence of qualitative predictor variables without creating dummy variables.
- Currently used widely and easy to interpret for non-expert.
- Closer to human decision making.
- Can be displayed graphically.
- Can handle missing data.
- Can handle multi-collinearity issue better than any other linear methods.

Weakness:

- When the distribution is given such as linear or quadratic between classification outcome and predictors, it will perform worse than other classification methods.
- Generally weaker predictive power.
- Suffer from instability.
- Easy to overfit.
- Need large sample sizes.

The tuning parameter would be α in cost-complexity pruning/weakest link pruning (cp : complexity parameter. Smaller cp leads bigger tree to learn more specific relations in the data, and it may result in overfitting). Also, we could also train minsplit (the minimum number of observation in a node for a split to take place) and minbucket (the minimum number of observation we can keep in terminal nodes).

- Main function and packages used : rpart package(rpart, prune, rpart.plot), caret package(train)

First, we created a baseline classification tree using gini index as a splitting criterion and set up the arguments in the list of control option. We set the minimum number of observation in a node for a split to take place(minsplit) as 5, complexity parameter(cp) as 0.0001, and maximum(maxdepth) depth as 5, and let tree grew. The reason why we set maximum depth as 5 was to make tree more interpretable and it turned out to be an appropriate maximum depth since almost all the terminal nodes in the tree object plot below were pure, and it gave enough information on predicting which types of individuals earned the income more than $50,000. The function that we used here was rpart in rpart package.

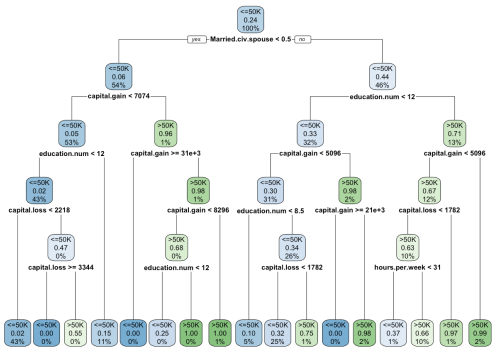


Figure 1: Random baseline classification tree

Each node in the graph showed the predicted class (<=50K or >50K income class), the predicted probability of >50K income class, and the percentage of observations in the node.

From the baseline classification tree model, we decided to build three tunned classification tree models by using different tuning methods. In order to see the overall performance of those models, we compared their AUC and test accuracy rate by using the optimal threshold that we picked from the training dataset. The three tuning methods and models that we used are following.

1. CP table from the model: From the baseline model, we picked the optimal tuning parameter, cp, which led to the smallest xerror (cross validation error = xerror * root note error) by using cptable since we wanted to minimize cross validation error for the tunned model. Then we pruned the tree using that cp, which is 0.0002603489. The function that we used in this step was prune in rpart package.

Click here - Fig 31 to see how the cross validation error changes as cp changes in appendix.

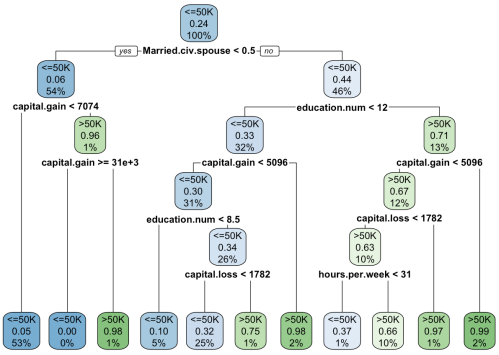


Figure 2: Classification tree pruned by cptable

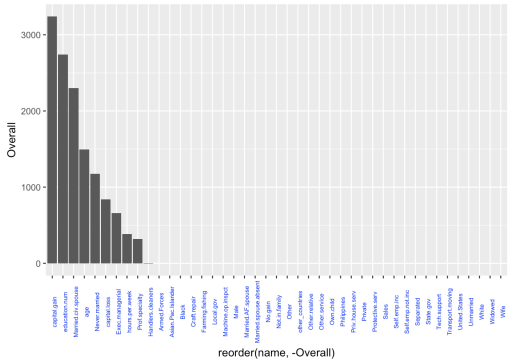


Figure 3: Visualization of variable importance statistics

The training accuracy of this model was 0.8509, and 'capital.gain,' 'education.num,' 'married.civ.spouse,' 'age,' 'never.married,' 'capital.loss,' and 'Exec.managenial' were important features as we could see in the visualization of variable importance statistics. As we could see that 'capital.gain' and 'education.num' were the most frequent splitting criterion in tree plot, it did make sense that they ranked as the first and the second important variable.

```
Confusion Matrix and Statistics

          Reference
Prediction <=50K  >50K
     <=50K 23559  1161
     >50K   3670  4012

               Accuracy : 0.8509
                 95% CI : (0.847, 0.8548)
    No Information Rate : 0.8403
    P-Value [Acc > NIR] : 0.0000000869
```

Figure 4: Confusion matrix for train dataset - tunned by the first method

2. Using the function train in caret package with gini index splitting criterion: We tunned the complexity parameter with caret function by setting the method as rpart, splitting criterion as gini index, and trained control method as repeated cv (10 folds, 3 times) in order to increase the accuracy of the model. In this case, the best tuning parameter(cp) was 0.00117157.

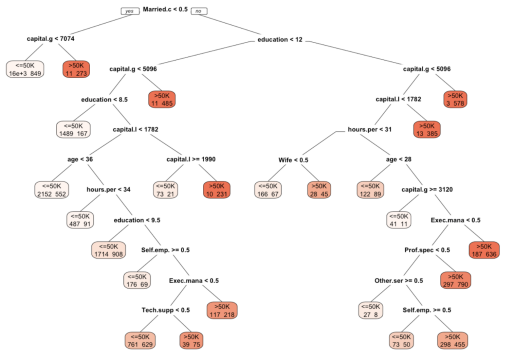


Figure 5: Classification tree pruned by caret function with gini index splitting criterion

```
Confusion Matrix and Statistics

          Reference
Prediction <=50K  >50K
     <=50K 23706  1014
     >50K   3511  4171

               Accuracy : 0.8603
                 95% CI : (0.8565, 0.8641)
    No Information Rate : 0.84
    P-Value [Acc > NIR] : < 0.00000000000000022
```

Figure 6: Confusion matrix for train dataset - tunned by the second method

Click here - Fig 32 for visualization of variable importance statistics in appendix

The training accuracy of this model was 0.8603, and the seven most important features in variable importance statistics were the same as the previous model.

3. Using function train in caret package with information gain splitting criterion: We tunned the complexity parameter by using caret function by setting the method as rpart, splitting criterion as information again (cross entropy), and

train control method as repeated cv (10 folds, 3 times) in order to increase the accuracy of the model. In this case, the best tuning parameter(cp) was 0.00117157, which was the same cp from the second method. However, the tree looked a bit different since we used different splitting criteria.

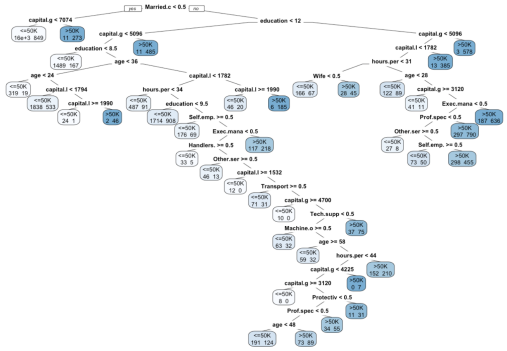


Figure 7: Classification tree pruned by caret function with information gain(cross entropy) splitting criterion

```
Confusion Matrix and Statistics

          Reference
Prediction <=50K  >50K
     <=50K 23440  1280
     >50K   3119  4563

               Accuracy : 0.8642
                 95% CI : (0.8605, 0.8679)
    No Information Rate : 0.8197
    P-Value [Acc > NIR] : < 0.00000000000000022
```

Figure 8: Confusion matrix for train dataset - tunned by the third method

Click here - Fig 33 for visualization of variable importance statistics in appendix

The training accuracy of this model was 0.8642, which was the highest among these three tunned models and, again, the seven most important features in variable importance statistics were the same as the previous models

Model selection

We implemented two ways to measure the performances of each model as we stated in the introduction: one way was comparing AUC, and the other way was comparing accuracy by the optimal threshold. We checked the accuracy rate of each model on train dataset by setting up the threshold from 0.001 to 0.999 with 0.001 increments and found out that 0.5 was always the cutoff, which gave us the maximum accuracy rate for our models. Therefore, we set 0.5 as the threshold for all models and calculated the accuracy rate on the test dataset.

Model Selection by AUC

In order to compare the performance of each model, we plotted ROC curve of four different models at the same time and measured AUC(area under the curve).

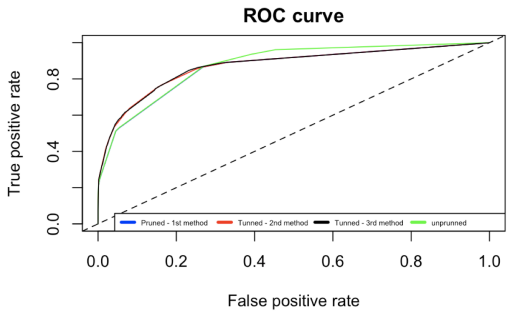


Figure 9: ROC curve of unpruned and three different tunned models

```
##   pruned.auc  gini.auc info.auc  tree.auc
## 1   0.858427 0.8718559 0.872279 0.8768653
```

Figure 10: Measured AUC from unpruned and three different tunned models

We could see that the unpruned tree's graph looked quite different than others while the other tunned models behaved quite similar.

As we stated in the introduction, we selected the model with the highest AUC, which was the unpruned tree (baseline model).

Model selection by Accuracy

We used the train dataset to pick a new threshold for each model by comparing training accuracy results from different thresholds. As we could see in the graphs below, all models achieved maximum accuracy at 0.5 threshold. Therefore, we chose 0.5 as the threshold for each model as we stated previously.

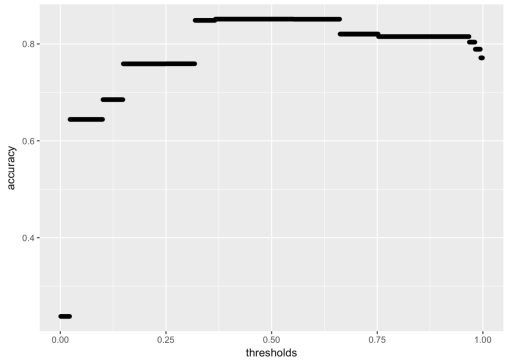


Figure 11: Accuracy by each threshold on train dataset - unpruned model

Click here - Fig 34 Accuracy by each threshold on train dataset in appendix - first tunned model

Click here - Fig 35 Accuracy by each threshold on train dataset in appendix - second tunned model

Click here - Fig 36 Accuracy by each threshold on train dataset in appendix - third tunned model

By using the chosen optimal thresholds from the test dataset, we got the test accuracy rates for each model. The accuracy rates by the optimal threshold from the test dataset for each model were following:

unpruned model (0.8517535) < first tunned model (0.8518153) < second tunned model (0.8602124) < third tunned model (0.8608916)

Since the third tunned model had the highest test accuracy rate, we selected the model, which was tunned by the train function in caret package that used information gain (cross entropy) as the splitting criterion.

In summary, we selected the unpruned tree model for AUC model selection and tunned model by information gain (cross entropy) splitting criterion for accuracy rate model selection.

2. Build a Bagged Tree

Bagged tree is originally stemmed from bootstrapping method, and this reduces the one of the biggest weakness of decision trees: high variance (thus improve accuracy over predictions with a single tree), by averating set of observations. To briefly explain what bagging is, it is the performed by taking repeated samples from the training data set: $\hat{f_{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)$ (ISL).

Our tuning parameter is the number of trees (as ISL says, this is not a critical parameter with bagging).

One weaknees is that it is harder to interpret the model.

Click here - Fig 37 to see an algorithm from APM

- Main function and packages used: randomForest package, mlr package (Mainly for tuning the parameters - makeResampleDesc, resample, makeLearner, train, makeTuneControlRandom, tuneParams, makeBaggingWrapper)

Bagged tree is a special type of random forest when mtry (number of variables selected at each split) is equal to the number of variables in the dataset. Therefore, it has some characteristics that random forest has, and one of them is invariant to monotonic transformations of individual features (variables). In other words, random forest only considers the range of one specific variable at each stage of splitting so we did not normalize variables since normalization will not change the results and information from it.

First, we used makeClassifTask in mlr package to create the train task and test task by setting up the option "positive" as ">50K" since we want to predict the individuals whose income was greater than $50,000. Then makeLearner in mlr package was used to make a bagging learner. We chose the class of learner as "classif.rpart" since our object was a classification of the binary response variable, set up gini index as a splitting criterion, the number of tree (ntree) as 50, and let other parameters as default. Function makeBaggingWrapper was used to set up the bagging algorithm which would grow 100 trees on randomized samples of data with replacement.

By setting up the 3 fold cross validation as validation strategy to check the performance, the baseline bagging model gave us 0.515 true positive rate, 0.0529 false positive rate, 0.485 false negative rate, 0.947 true negative rate, and 0.845 accuracy rate.

Then we made another random bagged learner by setting class of learner as "classif.randomForest" and mtry as 43 since there were 43 predictors in our dataset. This model gave us 0.622 true positive rate, 0.084 false positive rate, 0.378 false negative rate,

0.916 true negative rate, and 0.846 accuracy rate. Random forest used 0.5 as a cutoff by default. We noticed that false negative rate was quite high compared to false positive rate so we manually picked new cutoff as (0.55, 0.45). We increased the cutoff for negative classes (<=50K) and accordingly reduced it for positive classes (>50K) and trained the model again. This model gave us 0.665 true positive rate, 0.103 false positive rate, 0.335 false negative rate, 0.897 true negative rate, and 0.842 accuracy rate. The accuracy decreased but the difference between false negative and false positive rate also decreased.

Click here - Fig 38 to see how the training classification error changes as we increase the number of trees for untunned model in appendix

By getting the rough idea of how we should ntree from the graph, we tuned the hyperparameters ntree (the number of trees to grow) and nodesize (how many observations we want in the terminal nodes) by using tuneParams function in mlr package. We used three folds cross validation as validation strategy and set maxit as 5 in makeTuneControlRandom function as optimization technique. The reason why we set the three number of folds for CV was that it gave us almost the same accuracy rate of the tunned model and took less time than 5 or 10 folds of CV. We did not tune the depth of the tree since the nodesize was directly related to it. The optimal tunned parameters were 43 for nodesize and 68 for ntree. Also, mtry was 43.

We used these parameters for modeling and trained the model by using train function in mlr package.

Click here - Fig 39 to see the trained model in appendix.

Click here - Fig 40 to see how the training classification error changes as we increase the number of trees for tunned model in appendix

The seven most important variables from tuned model were 'capital.gain,' 'Married.civ.spouse,' 'education.num,' 'capital.loss,' 'age,' 'hours.per.week,' and 'Exec.managerial' from mean decrease in accuracy graph. These features were the same as the model from classification tree except the fact that 'hours.per.week' variable was added in bagged model instead of 'never.married' feature in classification tree. Only the order of the top seven important variables changed a bit in mean decrease gini variable importance graph.

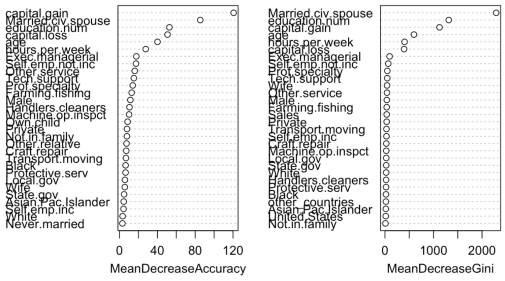


Figure 12: Visualization of variable importance statistics

The accuracy rate of this model on the train dataset was 0.8917.

Click here - Fig 41 to see the confusion matrix on train dataset in appendix

Click here - Fig 42 to see another visualization of prediction on train dataset in appendix

The accuracy rate of this model on the test dataset was 0.8598.

Click here - Fig 43 to see the confusion matrix on test dataset in appendix

Click here - Fig 44 to see another visualization of prediction on test dataset in appendix

Model selection

We implemented the same model selection method as we did in classification tree. However, when we checked the accuracy rate of each model on train dataset by setting up the threshold from 0.001 to 0.999 with 0.001 increments, we found out that optimal threshold from the train dataset was not always the best cutoff which gave us the maximum accuracy rate on the test dataset. The default cut off, which was 0.5 gave higher test accuracy rate than the optimal threshold that we picked from the train dataset. Therefore, we rather set 0.5 as the threshold for all models and calculated the accuracy rate on the test dataset.

Model Selection by AUC

In order to compare the performance of each model, we plotted ROC curve of different tuned and untuned models at the same time and measured AUC.

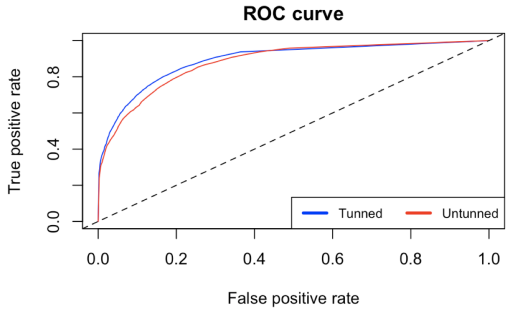


Figure 13: ROC curve of tuned and untuned bagged models

We could see that the tuned model's roc curve was more closed to the left top corner of the graph than untuned model's roc curve. The AUC of the untuned bagged model was 0.8817957 and tuned model's was 0.8942506. Therefore, we selected the model with the highest AUC which is the tuned bagged tree.

Model selection by Accuracy

The graph below was the untuned bagged model's training accuracy rate by each threshold.

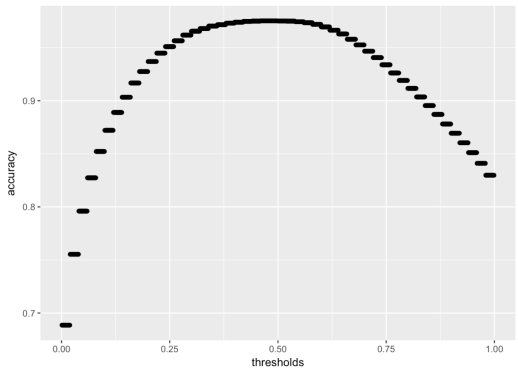


Figure 14: Accuracy by each threshold on train dataset - untuned bagged model

Click here - Fig 45 to see accuracy rate by each threshold on test dataset in appendix - tuned bagged model

The accuracy rates by the optimal threshold (0.5) from the test dataset for each model were following:

untuned bagged model(0.8449617) < tuned bagged model (0.8620647)

Since the tuned bagged model had higher test accuracy rate, we selected the tuned bagged model by the train function in mlr package that used gini index as the splitting criterion.

In summary, we selected the tuned bagged model for both AUC model selection and accuracy rate model selection.

3. Build a Random Forest

Random forest is another method improving bagged trees by decorrelating the trees (it is a combination of random variable selection and building decorrelated forest as we learned in lab 12). The biggest difference with a bagging tree is that we only consider some subsets (tuning is needed) of variables out of all. And, by doing it, we get less correlated trees, and thus, more likely to reduce variance when we average out uncorrelated trees, and thus, our model becomes more reliable.

So, if I say my subset size is p, where p is the number of variables from our data, it is the same as bagging tree.

Click here - Fig 46 to see an algorithm from APM in appendix

- Main function and packages used: randomForest package, mlr package(Mainly for tuning the parameters - makeClassifTask, makeResampleDesc, resample, makeLearner, train, makeTuneControlRandom, tuneParams)

First, we used makeClassifTask in mlr package to create the train task and test task by setting up the option "positive" as ">50K" as we explained in bagged tree part. Then makeLearner in mlr package was used to make a random forest learner. We

chose the class of learner as "classif.rpart" since our object was a classification of the binary response variable, set up gini index as the splitting criterion, the number of tree(ntree) as 50, and let other parameters as default.

By setting up the 3 fold cross validation as validation strategy to check the performance, the untuned model gave us 0.623 true positive rate, 0.0586 false positive rate, 0.377 false negative rate, 0.941 true negative rate, and 0.866 accuracy rate. Random forest uses a cutoff of 0.5 by default. We noticed that false negative rate was quite high compared to false positive rate like in bagged tree case. Therefore, we manually picked new cutoff as (0.53, 0.47). We increased the cutoff for negative classes (<=50K) and accordingly reduced it for positive classes (>50K) and trained the model again. This model gave us 0.645 true positive rate, 0.0657 false positive rate, 0.355 false negative rate, 0.934 true negative rate, and 0.866 accuracy rate. The accuracy stayed the same but the difference between false negative and false positive rate was decreased.

Click here - Fig 47 to see how the training classification error changes as we increase the number of trees for untuned model in appendix

By getting the rough idea of how we should set ntree from the graph, we tuned the hyperparameters which are ntree (number of trees to grow), nodesize (how many observations we want in the terminal nodes), and mtry (number of variables we should select at a node split) by using the tuneParams function in mlr package. We used three folds cross validation as validation strategy and set maxit as 5 in makeTuneControlRandom function as optimization technique like we did in bagged tree case. We did not tune the depth of the tree since the nodesize was directly related to it. The optimal tunned parameters were 14 for nodesize, 79 for ntree, and 8 for mtry.

We used these parameters for modeling and train the model by using train function in mlr package.

Click here - Fig 48 to see the trained model in appendix.

Click here - Fig 49 to see how the training classification error changes as we increase the number of trees for tuned model in appendix

The seven most important variables from tuned model were 'capital.gain,' 'education.num,' 'capital.loss,' 'age,' 'Married.civ.spouse,' 'hours.per.week,' and 'Exec.managerial.' They were the same as the model from bagged tree according to mean decrease accuracy graph below. However, the top seven important variable by mean decrease in gini index was different in random forest. Those were 'Married.civ.spouse,' 'capital.gain,' 'education.num,' 'age,' 'hours.per.week,' 'capital.loss,' 'Never.married.' The feature 'Exec.managerial' in mean decrease in accuracy was replaced by 'Never.married' in mean decrease in gini index variable importance plot.

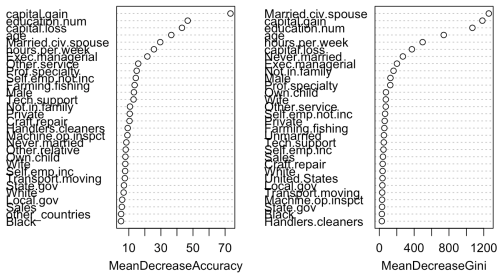


Figure 15: Visualization of variable importance statistics

The accuracy rate of this model on the train dataset was 0.8967.

Click here - Fig 50 to see the confusion matrix on train dataset in appendix.

Click here - Fig 51 to see another visualization of prediction on train dataset in appendix.

The accuracy rate of this model on the test dataset was 0.8644.

Click here - Fig 52 to see the confusion matrix on test dataset in appendix.

Click here - Fig 53 to see another visualization of prediction on test dataset in appendix.

Model selection

Like we stated in the previous bagged tree part, the default cut off (0.5) gave higher test accuracy rate than the optimal threshold that we picked from the train dataset in random forest as well. Therefore, we set 0.5 as the threshold for all models and calculated the accuracy rate on the test dataset.

Model Selection by AUC

In order to compare the performance of each model, we plotted ROC curve of different tuned and untuned models at the same time and measured AUC.

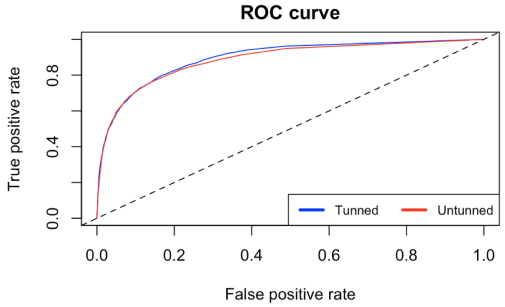


Figure 16: ROC curve of tuned and untuned random forest models

We could see that the tuned model's roc curve was more closed to the left top corner of the graph than untuned model's roc curve. The AUC of untuned bagged model was 0.8867613 and tuned model's was 0.8962369. Therefore, we selected the model with the highest AUC, which was the tuned random forest.

Model selection by Accuracy

The graph below was the untuned bagged model's training accuracy rate by each threshold.

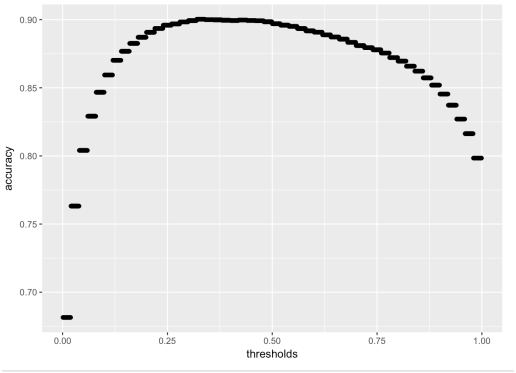


Figure 17: Accuracy by each threshold on train dataset - untuned random forest model

Click here - Fig 54 to see accuracy rate by each threshold on test dataset in appendix - tuned random forest model

The accuracy rates by the optimal threshold(0.5) from the test dataset for each model are following:

tuned random forest model(0.8647814) < untuned random forest model (0.866078)

Since the untuned random forest model had higher test accuracy rate, we selected the untuned bagged model by the train function in mlr package that used gini index as the splitting criterion.

In summary, we selected the tuned random forest model for AUC model selection and untuned random forest model for accuracy rate model selection.

4. Build a Boosted Tree (Extra)

The boosted tree is another method doing the similar way as bagging, but the differences are that each tree is grown sequentially based on information from previous trees and it does not use bootstrap sampling (fit on a modified version of the original data [basically, fit a decision tree to the residuals to update the residual where the current model is not doing good] as what ISL says). This boosting method particularly works well for classification tree than others such as LDA or KNN, because classification trees are a low bias and high variance method, this boosting helps lower the variance by making into weak learners by restricitng the tree depth.

Our tuning parameters are interaction.depth (how slowly to improve/learn the model), shrinkage parameter (parameter to allow more and different shaped tree but slowing down the process), and number of trees (use CV to tune).

The packages we used were "bst" and "gbm" for building boosted trees. The functions we used were "predict," "performance," "trainControl," "train," and "prediction."

Two different algorithms of boosted trees from APM: (our team is using **gbm**!!!) - click here - Fig 55 and 56

First of all, our group tried to plot gbm functions without using "train" function to get the basic ideas how the patterns of the test errors change based on different parameters we learned in the lab.

Here were the five different models we had tried first. In here, we used 0.5 as our cutoff (we are going to explain in detail why we chose 0.5 as our cutoff later) for the predicted probabilities. Our goal was to train the different models of boosted trees, find how the test and training error/accuracy rate changes based on parameters, find the variable important statistics, and attain ROC and AUC statistics (later, we are going to use the function "train" with either cross validation or bootstrap with 5 repeats of 10-fold CV, as the textbook APM recommends, and attain the optimal parameters, and do the similar steps).

Here we would not change "n.minobsinnode" (minimum of terminal node size), as we did not really cover this parameter in the class.

1. Here is a plot of the test error rate against the index/number of trees (index i indicates 10 * i number of trees) of boosted tree for when number of tree = 5000, interaction depth = 5, and shrinkage parameter = 0.001.

click here to see the a) test error v.s. number of trees b) train error v.s. number of trees - Fig 57 and 58

2. Here is a plot of the test error rate against the index/number of trees (index i indicates 10 * i number of trees) of boosted tree for when number of tree = 2000, interaction depth = 5, and shrinkage parameter = 0.001.

click here to see the a) test error v.s. number of trees b) train error v.s. number of trees - Fig 59 and 69

3. Here is a plot of the test error rate against the index/number of trees (index i indicates 10 * i number of trees) of boosted tree for when number of tree = 5000, interaction depth = 3, and shrinkage parameter = 0.001.

click here to see the a) test error v.s. number of trees b) train error v.s. number of trees - Fig 61 and 62

4. Here is a plot of the test error rate against the index/number of trees (index i indicates 10 * i number of trees) of boosted tree for when number of tree = 5000, interaction depth = 3, and shrinkage parameter = 0.2.

click here to see the a) test error v.s. number of trees b) train error v.s. number of trees - Fig 63 and 64

5. Here is a plot of the test error rate against the index/number of trees (index i indicates 10 * i number of trees) of boosted tree for when number of tree = 5000, interaction depth = 3, and shrinkage parameter = 0.1.

click here to see the a) test error v.s. number of trees b) train error v.s. number of trees - Fig 65 and 66

Explanations:

First of all, we wanted to mention that our threshold for getting these five train and test set errors was 0.5. We choose 0.5 because this was the default threshold, and also, we thought that false positve and false negative rates were equally bad (not one of them is worse/better than the other). Furthermore, it just took so much time (it takes around 20-30 minutes to run one for loop. . .) for us to try all different kinds fo thresholds, so we decided to just use one threshold, which is 0.5.

When we looked at the variable importance, we realized that there were 6 big important variables: "Married civ spouse," "education num," "age," "capital gain," "hours per week," and "capital loss." We found that the first one was always the "Married civ spouse"; however, the 2nd - 6th were changing based on which models I used. As you could see, we had tried different number of trees, interaction depth, and shrinkage parameters to find out how each of the parameters affect different features. And, by looking at the variable importance for our five different models, we could conclude that those six variables we mentioned above should be the most influential/important variables.

Another feature we realized was the similarity of train error rate and test error rate v.s. number of trees. When we just used the default shrinkage parameters (0.001), we found out the plots for test error rate v.s. number of trees and train error rate v.s. number of trees looked pretty similar. However, when we played around with shrinkage parameter, we found that these two plots look quite different. When we used the shrinkage parameters being equal to 0.1 or 0.2, the train error rates were strictly going down when number of trees increase; however, the test error rate was not strictly going down but bouncing up at some point. When shrinkage parameter is equal to 0.001 (default), both test error rate and train error plots strictly went down.

Another feature was learning rate. When we increased the shrinkage parameters to 0.1 or 0.2, both of test error and train error rates were low compared to the other three plots with shrinkage parameter being equal to 0.001. However, we learned in the lab that fast learner was not always a good thing, as the variance could increase up (and I found the paper saying that people usually recommend to use 0.1 for data sets with more than 10000 observations). As you could see from my test error plots with shrinkage parameters being equal to 0.1 or 0.2, the dots were not consistent, meaning that it showed more variability when shrinkage parameters increase (and we could even intuively tell that shrinkage parameter 0.1 looks better than the one with 0.2).

Another feature was how the plots looked different when the interaction depth (maximum tree depth or maximum nodes per tree) parameter changes. So, when we compared the first and third plots, we could easily find out that when the model with interaction.depth = 5 eventually achieve lower test and train error rates (especially show lower train error rates) comapred to the one with interaction.depth = 3. But, when the number of trees were small, the train and test error rates were similar.

Another feature was how the plots looked different when the number of trees were different. The model with number of trees being equal to 5000 showed lower train and test error rates (it makes sense when you see the plots, as when we use 2000 for number of trees, the plot was just cut off at index 200). However, as we learned in the class, too high numbers could cause over-fitting.

Again, when the shrinkage parameter was equal to 0.2, it seemed over-fitting and having high variance. (when the shrinkage was equal to 0.1, it slightly over-fitted, but it was not a serious issue)

And, after we understood some basic concepts of each parameter, we made some partial dependence plots for the first 6 important variables, below:

We only included one of them just for an example, and we plotted other fives in appendix.

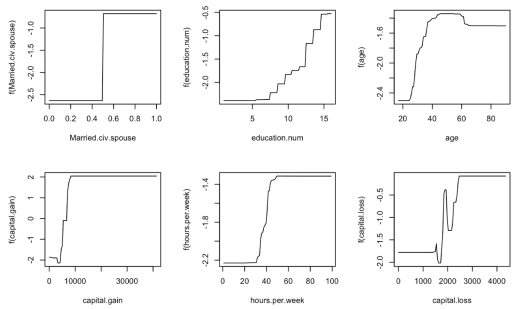


Figure 18: partial dependence model 1

click here to see dependce plots for other models - Fig 67 ~ 70

Explanations:

As the textbook ISL says, partial dependence plots show the marginal effect of the selected variables on the response after integrating out the other variables. In here, all the partial dependence plots of the variable "Married civ spouse" all look similar although you changed the number of trees, interaction depth, and shrinkage parameter. And, this made sense when we referred to the variable importance statistics. "Married civ spouse" was always the first one in variable importance statistics (the most influential), and we believed that whatever parameters we used, we would expect to see similar patterns.

So, we might expect income was increasing with "Married civ spouse," "Education num," and "Capital gain." The income might not strictly increase as the variables "age," "Hours per week," and "Capital loss" increase as they looked quite different as we changed the shrinkage parameters.

Before talking about ROC and AUC, we wanted to mention one thing! For shrinkage parameter = 0.001 (model 1, 2, and 3), both training and testing set errors kept decreasing as the number of trees grew, and for shrinkage parater being 0.1 (model 5) or 0.2 (model 4), training set errors kept decreasing but testing set errors bounced up at some point as the number of trees grew. So, for testing set ROC curve and AUC statistics, for model 1, 2, and 3, we decided to use 5000, 2000, and 5000 for number of trees, respectively, and for model 4 and 5, we decided to use 150 and 800 trees, respectively (please refer to the testing set errors v.s. number of trees). And, for training set ROC curve and AUC statistics, we used 5000, 2000, 5000, 5000, and 5000 as these were the number of trees that minimized the training set errors the most (please refer to the training set errors v.s. number of trees).

Also, remember that our train and test error rates were plotted when threshold = 0.5, and we decided to make the threshold = 0.5 for the following reasons. First, we do not put more weights on any of different types of errors. Second, it takes so much time if we try to check all testing and training sets errors (one threshold takes around 20 - 30 minutes).

Last but not least here were the testing set ROC curve and AUC statistics for our five models, below:

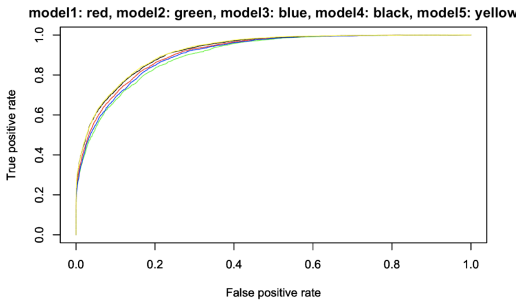


Figure 19: ROC testing gbm

The testing set AUCs for model 1 - 5 were 0.91392, 0.90268, 0.90865, 0.92097, and 0.92319, respectively.

As we could see, the model 5 (number of tree = 5000, interaction depth = 3, and shrinkage parameter = 0.1 - but used the 800 trees for predict function) had the highest AUC, meaning that the model 5 was the best classifier method for testing set.

And, here were the training set ROC curve and AUC statistics for our five models, below:

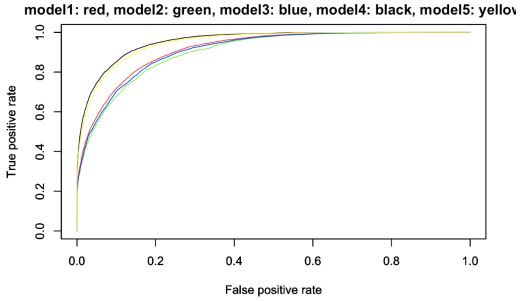


Figure 20: ROC training gbm

The training set AUCs for model 1 - 5 were 0.91611, 0.90325, 0.91001, 0.95562, and 0.94963, respectively.

Again, the model 4 (number of tree = 5000, interaction depth = 3, and shrinkage parameter = 0.2 - but used the 150 trees for predict function) had the highest AUC, meaning that the model 4 was the best classifier method for training set (remember that testing set AUC was what it matters!!!).

And, now, we are finally going to show off our optimal parameters!!!

We used a train function (with metric = "Accuracy") for boosted tree method, we used "gbm" for the method, as this was the one we used in the lab before. However, we might use some other types of boosting later on. click here to see a brief explanation of "gbm" method - Fig 71

The train function took a grid of parameter values and evaluated the performance using various flavors of cross-validation or the bootstrap, but we decided to use "repeated cv" to get the most optimal one. Again, we followed the package author's recommendaion: 5 repeats of 10-fold cross-validation (also what APM textbook uses).

The train function with gbm method fixed the the parameters "shrinkage" (0.1) and "n.minobsinnode" (10). We found that these two were fine (also I showed in my previous five examples that shrinkage = 0.1 is good).

So, we got the optimal tuning parameters (with the best accuracy rate and Kappa statistic - these two are performance measurements).

Here was a plot of the test error rate against the index/number of trees (index i indicates 10 * i number of trees) of boosted tree for when number of tree = 150, interaction depth = 3, and shrinkage parameter = 0.1 (min terminal node size = 10).

The top six important variables were the same as the previous five models we made.

Here was a testing set ROC curve with the previous five models:

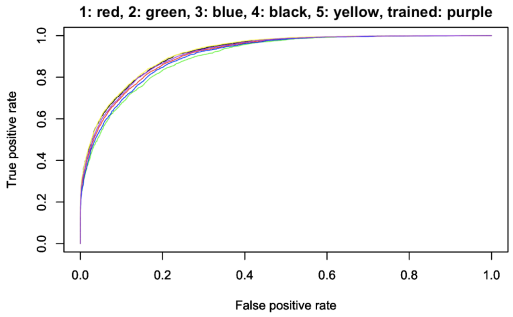


Figure 21: ROC testing combined

The AUC for testing set was 0.9177315.

click here to see roc curve on training - Fig 72

The AUC for training set was 0.9215941

So out of the six models (model 1 - 5 and trained model), the 5th model had the highest testing set AUC (it was definitely possible for the trained model had lower AUC because our train function had the metric = "Accuracy" not "AUC"). Again, as we said before, the fast learner was not always a good model as the variance could increase up, and that was why we said before we preferred the model with shrinkage parameter = 0.1 (model 5) over shrinkage parameter = 0.2 (model 4).

Now, we got the best threshold (the threshold with the smallest error rate) only with the train set, for each model. So, we drew the plots of training set error v.s. thresholds (we could not use testing set since it would be cheating), and found that it was better for us to keep the threshold = 0.5 (all closed to 0.5, and also tried to be consistent with other tree methods).

Again, we would keep the same number of trees as when we performed ROC curves and AUC statistics in previous steps.

We found that the model 1, 2, 3, 4, and 5 have 0.86299, 0.83502, 0.857187, 0.870029, and 0.871018, respectively. And, the model with trained function had 0.867004. We found it was definitely possible for the trained function to have lower accuracy rate in the testing set (definitely, trained function has the highest accuracy rate in the training set).

We asked Johnny whether the trained function model with metric = "Accuracy" could have lower accuracy than other methods. And, he cleared out my confusion by saying that "the best model chosen based on training data does not have the best performance in the test set. One reason could be that the distribution of the test set is very different from the distribution of training set (this situation is very common in time series data). Even if the training set and the test set came from the same distribution, it would be possible that the test error was worse just due to the nature of random sampling. Note that both CV errors and test errors were only estimates of predictive power and they were subject to uncertainties."

In conclusion, we would have the model 5 for our optimal model (the highest AUC and the highest accuracy rate in testing set) in boosted trees.

5. Model Selection

So, when we said "model selection," we could think of two definitions out of our heads.

1. Pick the best model (and when say say "best," we usually meant to be the model that has the best prediction accuracy) out of the different models.
2. Pick the best model out of the same models with bunch of different parameters.

However, mostly, in this section, we talked about the first one!!! (but, we did the second one for each method. So, basically, by doing #2 for classification decision tree, bagged tree, random forest, and boosted trees, we got the optimal/best parameters for each model - so called "inner model selction." And, then, by doing #1, we picked the best out of the best)

So, as we metioned before, we selected two models from each method for inner model selection: one model for the best AUC and the one for the best prediction accuracy rate in testing set. The ideal situation was when we had the same model achieving both conditions. Also, please click here to refer to Honorable Mention #3, to see our reason why we got both. Also, please be aware that we decided to use 0.5 instead of the optimal thresholds (as we explained before: 1) we found that the testing accuracy is higher/equal when we use 0.5 & 2) our optimal threshold is all closed to 0.5 anyway and we found using 0.5 can be better to be consistent & 3) we thought two types of errors are equally bad).

The classification tree and random forest had different models for the highest AUC and the best accuracy rate. And, the bagged and boosted had the same model for the highest AUC and the best accuracy rate (so, we have 2 models each from classification and random, and 1 model each for bagged and boosted - toal: 6 models).

First, we compared the four models we selected based on the highest AUC.

Here are the ROC curves, TPR v.s. TNR, and AUC for each model.

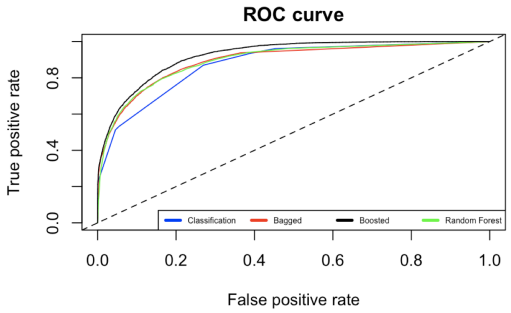


Figure 22: ROC Testing

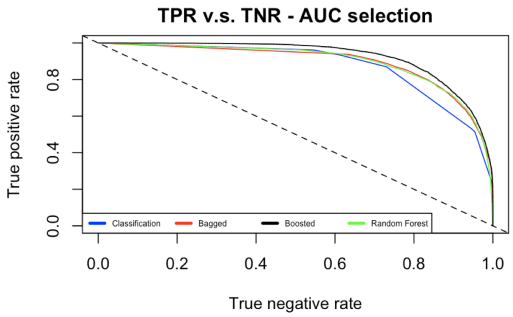


Figure 23: TNR v.s. TNR with the models based on AUC

AUCs were 0.87687, 0.89425, 0.89624, and 0.92319 for classficiation tree, bagged tree, random forest, and boosted tree, respectively.

And, now, we compared the four models we selected based on the highest testing accuracy rate.

Here are the TPR v.s. TNR and accuracy rates for each model (remeber that we do not need to include ROC or AUC here as we already categorized above).

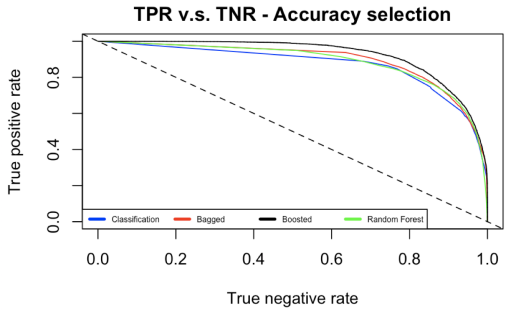


Figure 24: TNR v.s. TNR with the models based on testing set accuracy rate

Testin accuracy rates were 0.86089, 0.86206, 0.86608, and 0.87102 for classficiation tree, bagged tree, random forest, and boosted tree, respectively.

As you could see, whether we use "Best Accuracy rate" or "Highest AUC," we were having classficiation tree, bagged tree, random forest, and boosted tree from the worst to best classifier models.

Also, please refer to the knitted html for confusion matrix (we decided not to mention it as we already showed confusion matrix several times. And, we knew that the confusion matrix is a table with the fixed threshold).

Result

(summary of numerical analysis, interpretation, assumption check)

Classification Tree: We found that the unpruned classification tree gave us the highest AUC for the test dataset, and the pruned model by the information gain(cross entropy) splitting criterion gave us the highest accuracy rate. Also, we found that "capital.gain," "education.num," "married.civ.spouse," "age," "never.married," "capital.loss," and "Exec.managenial" were the most important variables. Our best model's AUC from the test dataset was 0.87687 and the accuracy rate was 0.86089.

Bagged Tree: We found that the tuned bagged tree gave us both the highest AUC and accuracy rate for the test dataset. Also, we found that "capital.gain," "married.civ.spouse," "education.num," "capital.loss," "age," "hours.per.week," and

"Exec.managenial" were the most important variables from both mean decrease in accuracy and gini graph. Our best model's AUC from the test dataset was 0.89425 and the accuracy rate was 0.86206.

Random Forest: We found that the tuned random forest gave us the highest AUC for the test dataset, and the untuned random forest gave us the highest accuracy rate. Also, we found that "capital.gain," "education.num," "capital.loss," "age," "married.civ.spouse," "hours.per.week," and "Exec.managenial" were the most important variables from mean decrease in accuracy graph. Our best model's AUC from the test dataset was 0.89624 and the accuracy rate was 0.86608.

Boosted Tree: For our data set, we found the boosted tree gave us the best testing set AUC and accuracy rate with the shrinkage parameter = 0.1 (this is the key parameter). And, we found "Married civ spouse," "education num," "age," "capital gain," "hours per week," and "capital loss" were the most important variables. Our best model output 0.92319 for the tesing set AUC and 0.87102 for the testing accuracy rate.

Conclusion

Based on our analysis respect to AUC and testing accuracy rate, our best supervised classifier (on testing set) was boosted tree (and following, random forest, bagged tree, and classification tree). We concluded that this outcome made sense based on the definitions. Boosted tree sequentially grew to improve the residual and took much more computing time than any other methods. Also, random forest was updated version of bagged trees, so it made sense for random forest being a better classifier than bagged tree.

How it can be further developed

First of all, as we mentioned above there might be better ways to deal with outliers, rahter than simplying removing them.

Second, for EDA, we could do better PCAs for pre-processing and understanding our data (but we could not go deeper because of time-constraint).

Third, for bagging, boosting, and random forest methods, we would love to try more different types if we had more time.

Fourth, for boosting, when we drew testing and training error v.s. number of tree plots, we hoped to draw many plots with different thresholds other than just being 0.5; however, since it took so much time to run it (20 - 30 minutes for one threshold), we could not run it.

Fifth, due to the limit of time, we might need to take "greedy" approach for each method to train and so called "optimal" parameters/models (on training set), but there was no guarantee that that this could give you the optimal answer (The tree methods were already taking "greedy" approach, so this was supposed to get "local optimmum," but we hoped to approach to "global optimum," but there was no guarantee to achieve "global optimum"). So, if we had more time and better computers, we wanted to train much more.

Appendix

Data

Here is how the define "fnlwgt" we could find from the website, below:

Description of fnlwgt (final weight)

The weights on the CPS files are controlled to independent estimates of the civilian noninstitutional population of the US. These are prepared monthly for us by Population Division here at the Census Bureau. We use 3 sets of controls. These are: 1. A single cell estimate of the population 16+ for each state. 2. Controls for Hispanic Origin by age and sex. 3. Controls by Race, age and sex. We use all three sets of controls in our weighting program and "rake" through them 6 times so that by the end we come back to all the controls we used. The term estimate refers to population totals derived from CPS by creating "weighted tallies" of any specified socio-economic characteristics of the population. People with similar demographic characteristics should have similar weights. There is one important caveat to remember about this statement. That is that since the CPS sample is actually a collection of 51 state samples, each with its own probability of selection, the statement only applies within state.

Last but not least, here is the information from the website (link for the dataset), below:

- Randomly split into training (2/3) and test (1/3).
- 48842 instances, mix of continuous and discrete (train=32561, test=16281)
- 45222 if instances with unknown values are removed (train=30162, test=15060)
- Duplicate or conflicting instances : 6
- Class probabilities for `adult.all` file:
  - Probability for the label `>50K`: 23.93% / 24.78% (without unknowns)
  - Probability for the label `<=50K`: 76.07% / 75.22% (without unknowns) Error Accuracy reported as follows, after removal of unknowns from | train/test sets): | C4.5 : 84.46+-0.30 | Naive-Bayes: 83.88+-0.30 | NBTree : 85.90+-0.28

EDA

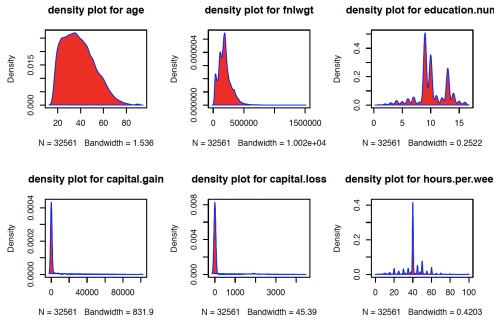


Figure 25: Density plots for continuous variables

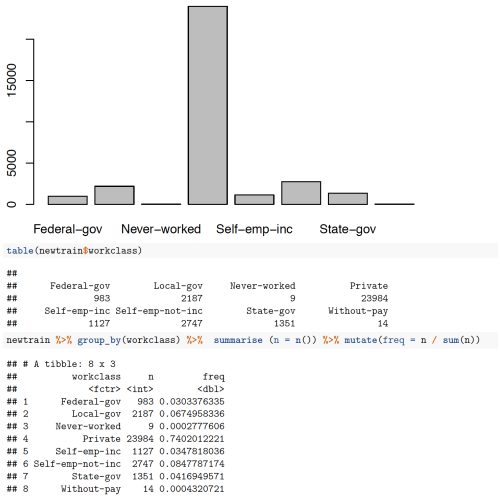
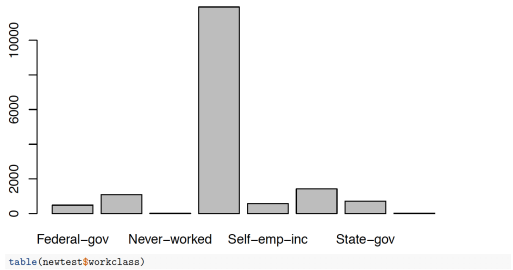


Figure 26: Workclass variable for train




```
table(newtest$workclass)

##
##      Federal-gov        Local-gov     Never-worked          Private
##              480             1089                3            11919
##     Self-emp-inc Self-emp-not-inc        State-gov      Without-pay
##              570             1421              707                7
```

```
newtest %>% group_by(workclass) %>%  summarise (n = n()) %>% mutate(freq = n / sum(n))

## # A tibble: 8 x 3
##          workclass     n        freq
##              <fctr> <int>       <dbl>
## 1      Federal-gov   480 0.0296369474
## 2        Local-gov  1089 0.0672388244
## 3     Never-worked     3 0.0001852309
## 4          Private 11919 0.7359224500
## 5     Self-emp-inc   570 0.0351938750
## 6 Self-emp-not-inc  1421 0.0877377130
## 7        State-gov   707 0.0436527538
## 8      Without-pay     7 0.0004322055
```

Figure 27: Workclass variable for test

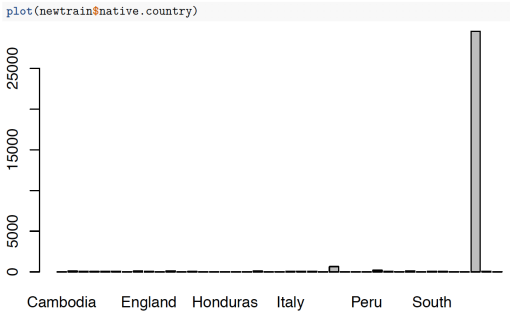


Figure 28: Native country variable

```
> newtest %>% group_by(education.num) %>% summarise (n = n()) %>% mutate(freq = n / sum(n))
# A tibble: 16 x 3
   education.num     n        freq
           <int> <int>       <dbl>
 1             1    32 0.001975796
 2             2    79 0.004877748
 3             3   175 0.010805137
 4             4   309 0.019078785
 5             5   242 0.014941961
 6             6   456 0.028155100
 7             7   637 0.039330699
 8             8   224 0.013830575
 9             9  5272 0.325512472
10            10  3583 0.221227464
11            11   677 0.041800445
12            12   534 0.032971104
13            13  2648 0.163497160
14            14   922 0.056927636
15            15   236 0.014571499
16            16   170 0.010496419
```

Figure 29: Education.num variable

```
newtest %>% group_by(education) %>%  summarise (n = n()) %>% mutate(freq = n / sum(n))

## # A tibble: 16 x 3
##       education     n        freq
##          <fctr> <int>       <dbl>
##  1         10th   456 0.028155100
##  2         11th   637 0.039330699
##  3         12th   224 0.013830575
##  4      1st-4th    79 0.004877748
##  5      5th-6th   175 0.010805137
##  6      7th-8th   309 0.019078785
##  7          9th   242 0.014941961
##  8    Assoc-acdm   534 0.032971104
##  9     Assoc-voc   677 0.041800445
## 10     Bachelors  2648 0.163497160
## 11     Doctorate   170 0.010496419
## 12       HS-grad  5272 0.325512472
## 13       Masters   922 0.056927636
## 14     Preschool    32 0.001975796
## 15  Prof-school   236 0.014571499
## 16 Some-college  3583 0.221227464
```

Figure 30: Education variable

Classification Tree

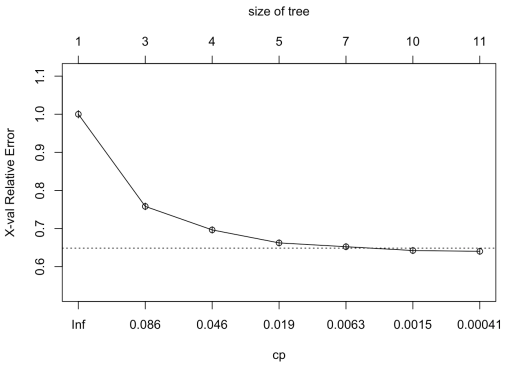


Figure 31: Classification

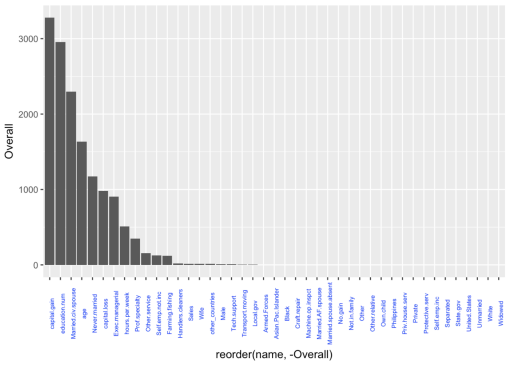


Figure 32: Classification

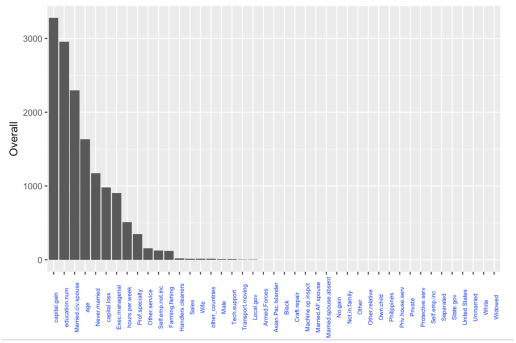


Figure 33: Classification

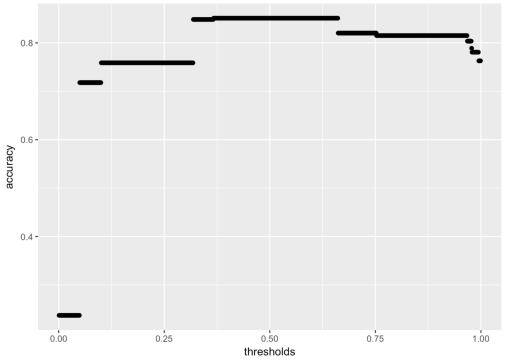


Figure 34: Accuracy by each threshold on train dataset - first tuned model

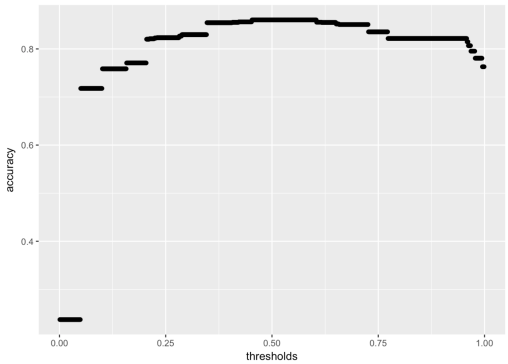


Figure 35: Accuracy by each threshold on train dataset - second tuned model

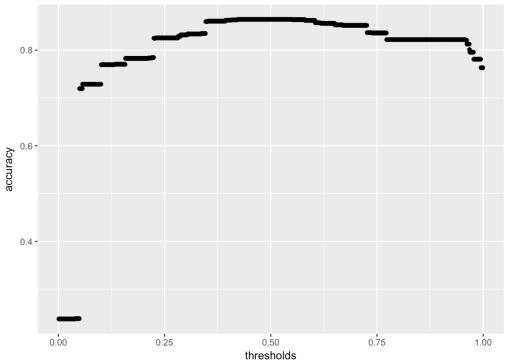


Figure 36: Accuracy by each threshold on train dataset - third tuned model

Bagged Tree

1 **for** *i = 1 to m* **do**
2 | Generate a bootstrap sample of the original data
3 | Train an unpruned tree model on this sample
4 **end**

Algorithm 8.1: Bagging

Figure 37: Bagging

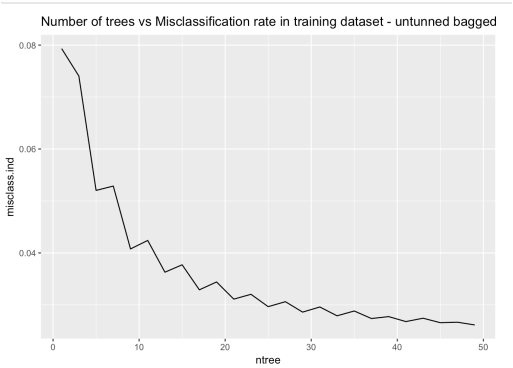


Figure 38: Number of trees VS training classification error

```
##
## Call:
##  randomForest(formula = f, data = data, classwt = classwt, cutoff = cutoff,      ntree = 68L, mtry = 43, impor
tance = TRUE, nodesize = 43L)
##                Type of random forest: classification
##                      Number of trees: 68
## No. of variables tried at each split: 43
##
##         OOB estimate of  error rate: 13.66%
## Confusion matrix:
##       <=50K >50K class.error
## <=50K 22960 1760  0.07119741
## >50K   2666 5016  0.34704504
```

Figure 39: Trained bagged model by tuned hyperparameters

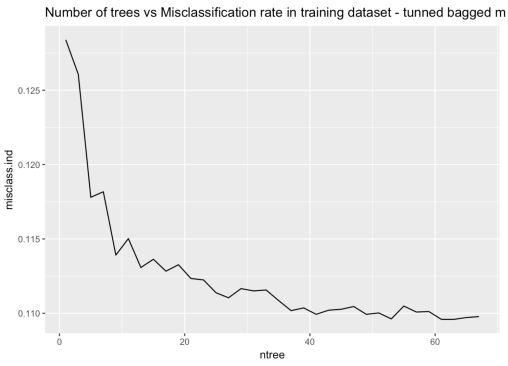


Figure 40: Number of trees VS training classification error

```
Confusion Matrix and Statistics

          Reference
Prediction <=50K  >50K
     <=50K 23414  2203
     >50K   1306  5479

               Accuracy : 0.8917
                 95% CI : (0.8883, 0.8951)
    No Information Rate : 0.7629
    P-Value [Acc > NIR] : < 0.00000000000000022
```

Figure 41: Confusion matrix on train dataset - tuned bagged tree

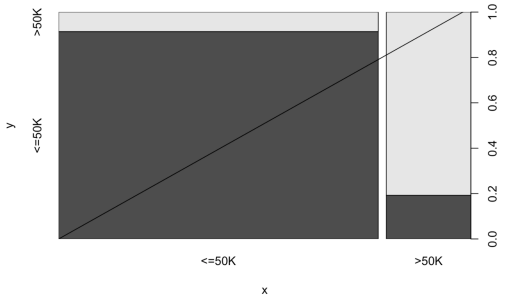


Figure 42: Proportion of predicted income on train dataset - white: >50K, black : <=50K

```
Confusion Matrix and Statistics

          Reference
Prediction <=50K  >50K
     <=50K 11504  1340
     >50K    931  2421

               Accuracy : 0.8598
                 95% CI : (0.8543, 0.8651)
    No Information Rate : 0.7678
    P-Value [Acc > NIR] : < 0.00000000000000022
```

Figure 43: Confusion matrix on test dataset - tuned bagged tree

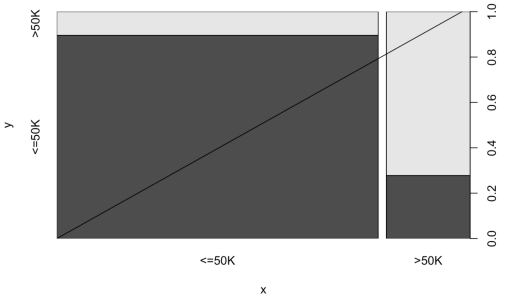


Figure 44: Proportion of predicted income on test dataset - white: >50K, black : <=50K

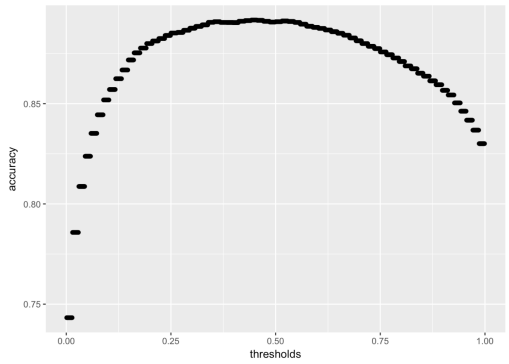


Figure 45: Accuracy by each threshold on train dataset - tuned bagged model

Random Forest

1 Select the number of models to build, m
2 **for** $i = 1$ *to* m **do**
3 | Generate a bootstrap sample of the original data
4 | Train a tree model on this sample
5 | **for** *each split* **do**
6 | | Randomly select k ($< P$) of the original predictors
7 | | Select the best predictor among the k predictors and partition the data
8 | **end**
9 | Use typical tree model stopping criteria to determine when a tree is complete (but do not prune)
10 **end**

Algorithm 8.2: Basic Random Forests

Figure 46: Random Forest Algorithm

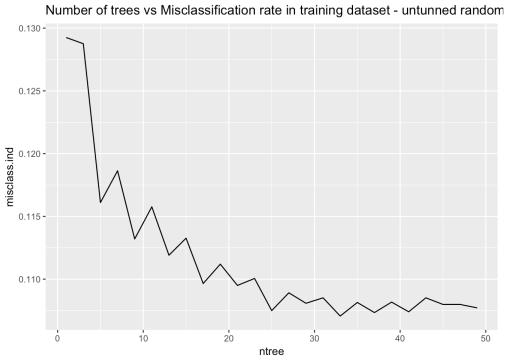


Figure 47: Number of trees VS training classification error - untuned random forest model

```
##
## Call:
##  randomForest(formula = f, data = data, classwt = classwt, cutoff = cutoff,      ntree = 79L, importance = TRU
E, mtry = 8L, nodesize = 14L)
##                Type of random forest: classification
##                      Number of trees: 79
## No. of variables tried at each split: 8
##
##         OOB estimate of  error rate: 13.47%
## Confusion matrix:
##       <=50K >50K class.error
## <=50K 23063 1657  0.06703074
## >50K   2709 4973  0.35264254
```

Figure 48: Trained random forest model by tuned hyperparameters

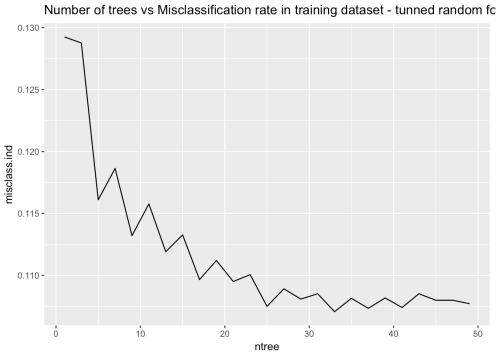


Figure 49: Number of trees VS training classification error

```
Confusion Matrix and Statistics

          Reference
Prediction <=50K  >50K
     <=50K 23561  2189
     >50K   1159  5493

               Accuracy : 0.8967
                 95% CI : (0.8933, 0.9)
    No Information Rate : 0.7629
    P-Value [Acc > NIR] : < 0.00000000000000022
```

Figure 50: Confusion matrix on train dataset - tuned random forest

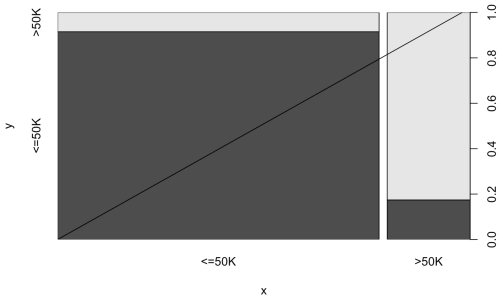


Figure 51: Proportion of predicted income on train dataset - white: >50K, black : <=50K

```
Confusion Matrix and Statistics

          Reference
Prediction <=50K  >50K
     <=50K 11601  1362
     >50K    834  2399

               Accuracy : 0.8644
                 95% CI : (0.859, 0.8696)
    No Information Rate : 0.7678
    P-Value [Acc > NIR] : < 0.00000000000000022
```

Figure 52: Confusion matrix on test dataset - tuned random forest

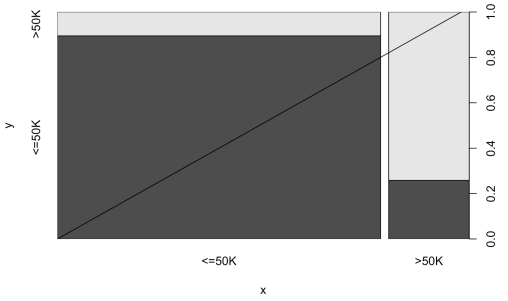


Figure 53: Proportion of predicted income on test dataset - white: >50K, black : <=50K

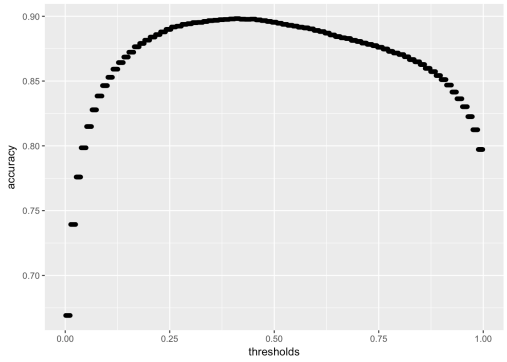


Figure 54: Accuracy by each threshold on train dataset - tuned random forest

Boosted Tree

1 Let one class be represented with a value of +1 and the other with a value of -1
2 Let each sample have the same starting weight $(1/n)$
3 **for** k = *1 to* K **do**
4 Fit a weak classifier using the weighted samples and compute the kth model's misclassification error (err_k)
5 Compute the kth stage value as $\ln\left(\left(1-err_k\right)/err_k\right)$.
6 Update the sample weights giving more weight to incorrectly predicted samples and less weight to correctly predicted samples
7 **end**
8 Compute the boosted classifier's prediction for each sample by multiplying the kth stage value by the kth model prediction and adding these quantities across k. If this sum is positive, then classify the sample in the +1 class, otherwise the -1 class.

Algorithm 14.2: AdaBoost algorithm for two-class problems

Figure 55: AdaBoost Algorithm

1 Initialized all predictions to the sample log-odds: $f_i^{(0)} = \log\frac{\widehat{p}}{1-\widehat{p}}$.
2 **for** *iteration* $j = 1 \ldots M$ **do**
3 Compute the residual (i.e. gradient) $z_i = y_i - \widehat{p}_i$
4 Randomly sample the training data
5 Train a tree model on the random subset using the residuals as the outcome
6 Compute the terminal node estimates of the Pearson residuals: $r_i = \frac{1/n\sum_i^n (y_i - \widehat{p}_i)}{1/n\sum_i^n \widehat{p}_i(1-\widehat{p}_i)}$
7 Update the current model using $f_i = f_i + \lambda f_i^{(j)}$
8 **end**

Algorithm 14.3: Simple gradient boosting for classification (2-class)

Figure 56: Simple Gradient Boosting

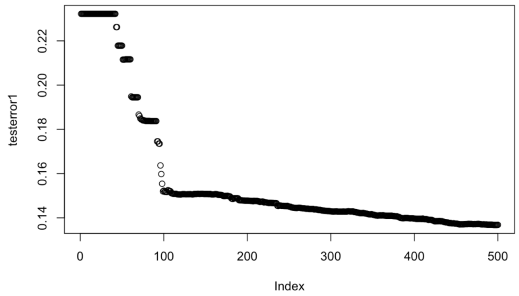


Figure 57: Test error rate v.s. number of trees model 1

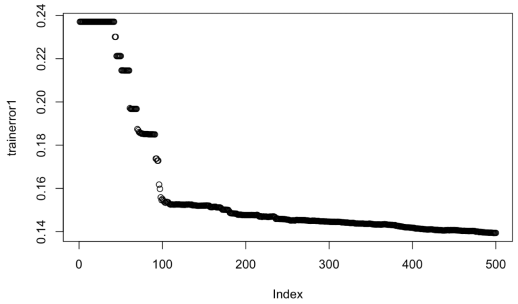


Figure 58: Train error rate v.s. number of trees model 1

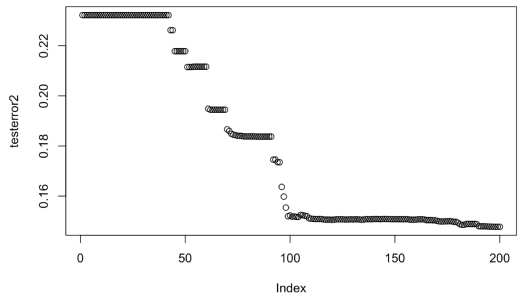


Figure 59: Test error rate v.s. number of trees model 2

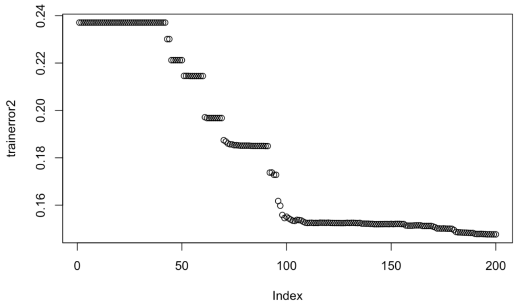


Figure 60: Train error rate v.s. number of trees model 2

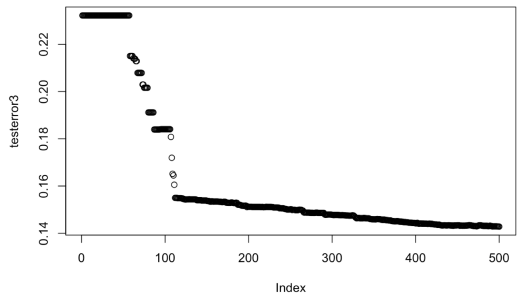


Figure 61: Test error rate v.s. number of trees model 3

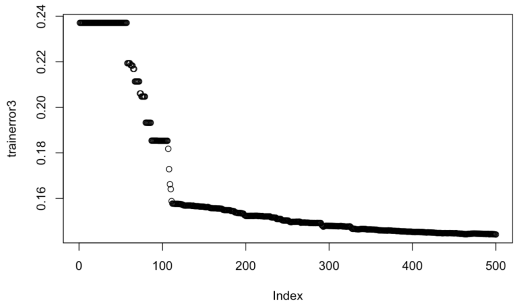


Figure 62: Train error rate v.s. number of trees model 3

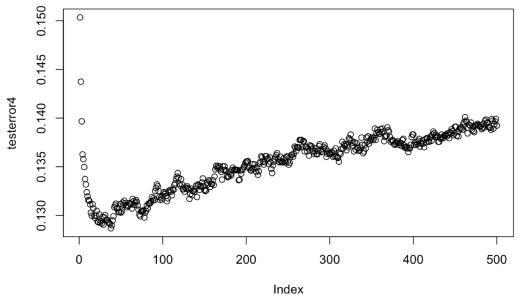


Figure 63: Test error rate v.s. number of trees model 4

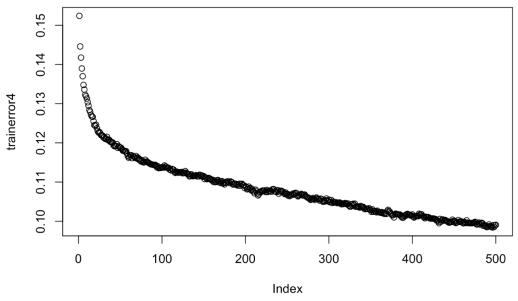


Figure 64: Train error rate v.s. number of trees model 4

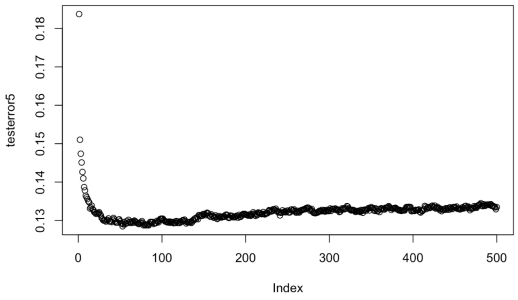


Figure 65: Test error rate v.s. number of trees model 5

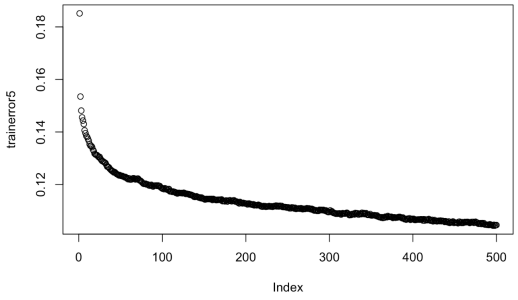


Figure 66: Train error rate v.s. number of trees model 5

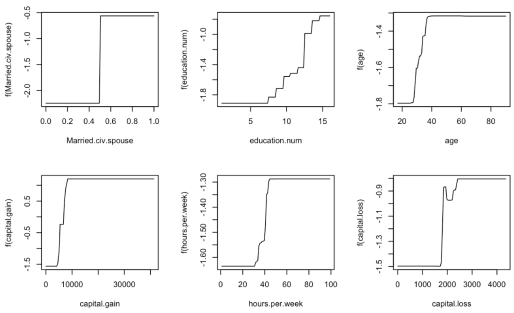


Figure 67: Partial dependence model 2

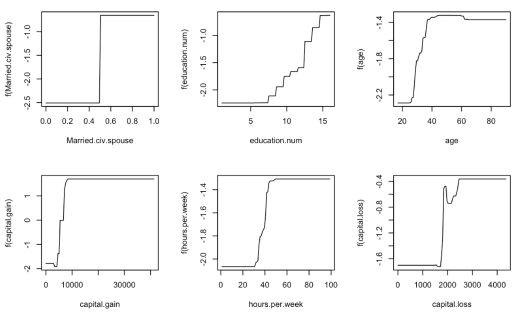


Figure 68: Partial dependence model 3

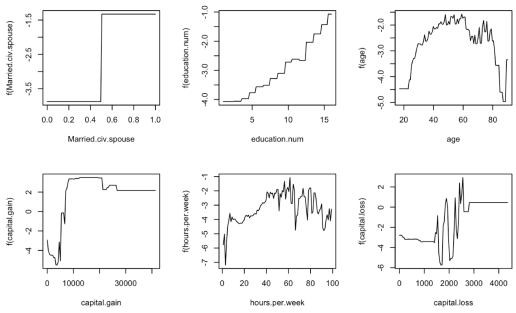


Figure 69: Partial dependence model 4

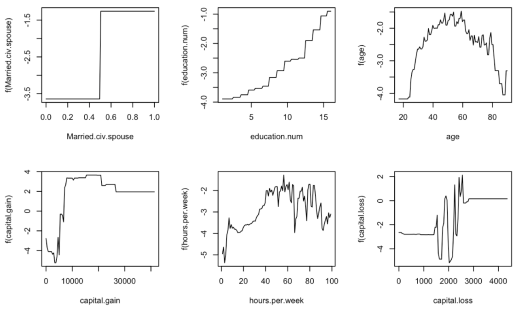


Figure 70: Partial dependence model 5

Stochastic Gradient Boosting

```
method = 'gbm'
```

Type: Regression, Classification

Tuning parameters:

- `n.trees` (# Boosting Iterations)
- `interaction.depth` (Max Tree Depth)
- `shrinkage` (Shrinkage)
- `n.minobsinnode` (Min. Terminal Node Size)

Required packages: `gbm` , `plyr`

Figure 71: Gbm method from training function

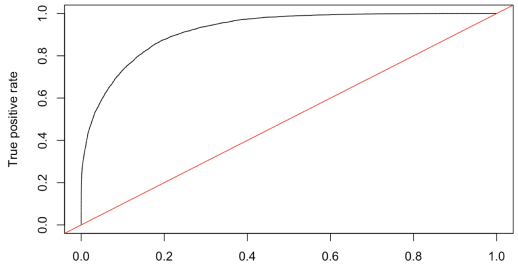


Figure 72: ROC training gbm

Reference

link for dataset

link for dictionary

James, Gareth, et al. An introduction to statistical learning with applications in R. Springer, 2017.

Kuhn, Max, and Kjell Johnson. Applied predictive modeling. Springer, 2016.

link for missing values fix

link for categorical variable outliers

gini, entropy, and misclassification-rate

Train R function

link to save plots

tuning paper

Image in rmarkdown

Image in rmarkdown2