MA429 Mock Project: Group 8

School:	The London School of Economics and Political Science	
Course:	MA429 Algorithmic Techniques for Data Mining	
Seminar:	1 (Monday 14:00 to 15:30)	
Professor:	Gregory Sorkin	
Date Submitted:	12 March 2018	
Team Members:	**Traci Lim**	201726385
	Yi Luo	201704053
	William Skinner	201749922

Prediction of Low Income Levels

From 1994 United States Census Data

Table of Contents

Table of Figures

Background and Aims

This project was performed as a mock project for the MA429 Algorithmic Techniques for Data Mining course at the London School of Economics. Our main objective was to gain experience with applying data mining techniques to a real dataset.

The dataset used for this project is from the 1994 United States census data available from the University of California, Irvine Machine Learning Repository (Kohavi & Becker, 1996). The data contains demographic and employment related features. It has 48,842 instances, each described by 15 features: *age*, *workclass*, *fnlwgt*, *education*, *education.num*, *marital.status*, *occupation*, *relationship*, *race*, *sex*, *capital.gain*, *capital.loss*, *hours.per.week*, *native.country*, and *annual.income*. In the dataset, of the 14 features that are not annual income, 6 features are numerical, while 8 features are categorical.

Governments interested in engaging in affirmative action to benefit lower income populations may find it useful to have a model that predicts whether people with certain characteristics would have earned less than 50 thousand dollars per year in 1994. This model would enable them to input the same characteristics for people today to see whether the income level they would likely have had in 1994 would have been less than 50 thousand dollars per year which could be used to evaluate and substantiate any findings on social welfare, say, improvement in living conditions or individual growth in paygrade.

The aim of this project is to provide such a model that predicts whether income in the United States is *less* than fifty thousand dollars per year.

Executive Summary

We first used univariate and multivariate analysis to examine the dataset, revealing the class imbalance problem. Then, to prepare the data for model building, we created four separate data sets to evaluate the effectiveness of three different resampling approaches to solve the class imbalance problem. To reach the best predicting power, we experimented with several learning methods, and we also normalized numerical variables when appropriate for the method. Considering both the success metrics and running time, we narrowed down to 5 main methods: *logistic regression*, *linear discriminant analysis* (LDA), *classification and regression trees* (CART), *tree boosting* (C5.0) and *Stacking*. Among them, the best results were produced by *Stacking*, which combined the strengths of three of the methods that we used, LDA, CART, and C5.0. The *Kappa* score for Stacking was 0.6129, with *sensitivity* 0.9437, *specificity* 0.6289 and AUROC (Area Under Receiver Operator Characteristic) of 0.920. The predicting ability of our model has the potential application to improve the efficiency of a social welfare system where the government needs to allocate subsidies to those in the greatest need.

Methodology

The methodology we followed is shown in Figure 1 below. To have the flexibility to make our own training and test data split we decided to merge the Adults.data and Adults.test data files. We then randomly split them into 70% Training set and 30% Test set and recorded which instances belong to each set for later splitting following preprocessing. We performed the pre-analysis on the unprocessed training set so that our decisions for preprocessing were not dependent on the test data. Next, we applied the same preprocessing steps to the entire dataset so that we only had to perform each preprocessing step once. Following preprocessing, we again split the data into 70% Training and 30% Test using the same index for splitting as before.

To compare the effects of resampling approaches, we created four different training datasets: without resampling, with oversampling, with undersampling, and with both. We trained our models on each of them to evaluate which resampling approach was best for that method based on our chosen performance metrics. For each method, we used 10-fold cross validation for model tuning.

Finally, we compared the best classifiers for each model using our chosen performance metrics computed with the testing set to pick the best model.

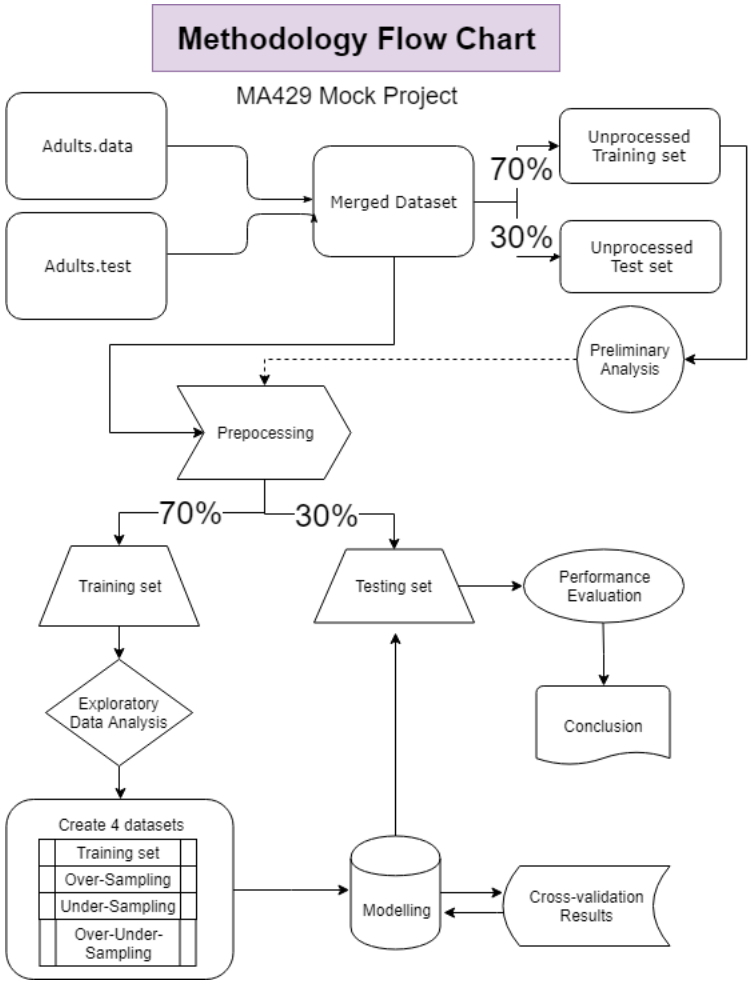

Figure 1: Methodology Flow Chart

Preliminary Analysis

In this following section, we briefly describe our analysis of individual features and of correlations between features which we used to identify how to preprocess the dataset to prepare it for our model building. We performed this analysis on the unprocessed training set to ensure that our preprocessing decisions were not influenced by the test data.

Univariate Analysis

First, we analyzed both numerical and categorical features individually to identify any unusual distributions or low frequency categories. We also looked at the difference between the distributions of the instances of each numerical feature for high and low income levels to identify any features we should exclude.

Numerical Features

We generated histograms, kernel density plots, and boxplots for each of the numerical features. These are shown in Appendix A, Figures 2 to 8. The "*age*" feature looks the closest to having a normal distribution, but it still had a clear right-skew. To statistically verify this skew, we conducted *the Cramer-von Mises test* and *the Anderson-Darling test* for normality which both had p-values less than 0.00001% so there is strong statistical evidence that age does not have a normal distribution. Looking at the boxplots in Figure 8 in Appendix A, we found visually apparent differences in distribution between more and less than 50k income for age, years of education, and hours worked per week. The weight feature, "*fnlwgt*", did not appear to be distinguishable between the two income levels and so is not very useful for prediction. The *capital.gain* and *capital.loss* features were dominated by zeros, however, the non-zero values do appear to be distinguishable between the two income levels.

Categorical Features

We generated bar plots for each of the categorical features. These are shown in Appendix A, Figures 9 to 13. We looked at the bar plots and decided that some levels, such as countries other than the US, have very low frequencies and could therefore be combined in the preprocessing stage.

For the income levels, 76% were less than 50k and only 24% were greater than 50k. This indicated that we were facing a class imbalance problem. This can be seen visually in Figure 13 in Appendix A.

The class imbalance problem is known to heavily bias models because they focus more on the classification of the majority sample while ignoring or misclassifying the minority sample. It causes a classifier to perform sub-optimally.

Multivariate Analysis

Using highly correlated features to build a model can result in detrimental redundancy. Therefore, we examined the relationship between features using multivariate analysis to decide which features to use as predictors.

Relationships between numerical features

To illustrate the correlation among numerical features, we created the correlogram plot shown in Figure 14. Correlations with crosses in Figure 14 are insignificant with p-value > 0.05. We observe in the correlogram that there are no strong correlations between numerical features, which is good because a feature highly correlated with another feature doesn't add much to predictive power.

Relationships between categorical features

In order to reveal the relationship between categorical features, we used *Goodman and Kruskal's tau*. Figure 15 in Appendix A illustrates the tau value between each categorical feature and each other categorical feature. The tau measure is based on the fraction of variability in the categorical variable y that can be explained by the categorical variable x. The tau measure is asymmetric, meaning that the fraction of variability in x that is explainable by variations in y may be very different from the variability in y that is explainable by variations in x (Pearson, 2016). In this chart, for example, while 0.01 of the variability of *workclass* can be explained by *education*, 0.02 of the variation in education can be explained by *workclass*. From the matrix, the tau value from *marital.status* to relationship is 0.59, meaning having knowledge of *marital.status* helps a lot in predicting relationship. Checking if there is a correlation between *marital.status* and relationship features using the chi-square test indicates that there is a correlation with a p-value less than 0.001%.

Preprocessing

For each categorical variable we looked at how many levels that feature had and the frequencies of each level. For some features there were many levels with low frequencies and others with high frequencies. For example, the feature *native.country* had 42 levels with "United States" significantly outweighing the rest of the levels. To make the weakly represented countries' information more distinguishable from underlying noise, it makes sense to reduce the number of levels by combining all levels with sparse instances into one level "*others*". Although this does result in a loss of information, it may be offset by the benefits of having more even level frequencies and a simpler model less sensitive to overfitting. Combining low frequency levels also improves running time. We combined levels for several other features as well. These combinations are summarized in Table 1. The combinations were made intuitively. An extension to this project could be to analyze and justify each combination.

Table 1: Combination of low-frequency levels of categorical features

Adjusted Feature	**Levels Combined**
workclass	Self-emp-inc, Self-emp-not-inc→Self-employed Never-worked, Without-pay→Not-working Local-gov, State-gov→Non-federal-gov
education	10th,11th,12th,1st-4th,5th-6th,7th-8th,9th→Dropout Assoc-acdm, Assoc-voc→Associates HS-grad, Some-college→HSGrad Doctorate, Prof-school→PhD
marital.status	Married-spouse-absent, Separated→Not-w-spouse Married-AF-spouse, Married-civ-spouse→Married
occupation	Craft-repair, Farming-fishing, Handlers-cleaners, Machine-op-inspct, Transport-moving→Blue-collar Other-service, Priv-house-serv→Service Protective-serv, Tech-support→Other-occ
race	Amer-Indian-Eskimo, Asian-Pac-Islander, Other→Other
native.country	Mexico, ?, Philippines, Germany, …, Puerto-Rico→Other

Histograms of the frequencies of each categorical variable level, before the low-frequency levels were lumped together, are shown in Figures 9 to 13 in Appendix A.

Normalizing a numerical variable can improve the performance of machine learning classifiers. Rather than simply subtracting the mean and dividing by the standard deviation, a skewed feature

can be more effectively normalized by applying a *Box-Cox transformation* to also help the data points to be spread closer to a normal distribution (Scibilia, 2015).

Missing values for categorical features were already encoded as question marks in the source data and treated as their own levels. We treated the missing value levels in the same way as any other level. Another approach could be to use an imputation technique such as multiple imputation to fill-in the missing values. Multiple Imputation by Chained Equations can be implemented using the *MICE* package in R. This is something that could be done to extend our work. There were no missing values in the numerical features.

We did not perform any special operations to handle outliers.

Choice of Performance Metrics: *Kappa statistic* and *Sensitivity*

Accuracy would be a poor choice of evaluation metric because the results would be misleading since minority classes hold minimal effect on overall accuracy (Longadge, Dongre, & Malik, 2013).

Other than being interested in whether the classification is correct or not, we might also be concerned with the true positive rate, also known as *sensitivity*. It measures the proportion of positives that are correctly identified as such.

Depending on the motivation of the project, say, if we want to pick out families that have less than 50k annual income, so as to give higher subsidies, some emphasis can be placed on optimising our models based on the true positive rate. Since not giving higher subsidies to some lower income families can have a more detrimental welfare impact than giving subsidies to some misclassified families that have more than 50k annual income, true negative rate is not as important as true positive rate. In other words, we care about *sensitivity* more than *specificity* when evaluating our models.

The *Kappa statistic* is a normalised statistic that can be used as a metric for classification algorithms. To put it simply, it measures how much better the model is compared to random guessing relative to the perfect classification. It is a very useful measure to use on problems that have an imbalance in the classes (Standard Wisdom LLC., 2011).

Therefore, we selected the *Kappa statistic* and *sensitivity* as performance evaluators.

Resampling

Resampling involves drawing repeated samples from the original dataset. In this case, we resample the dataset to strategically bring the distribution of the data set closer to the optimal distribution by removing samples from the majority class and/or adding samples to the minority class.

Because the optimal distribution of the data is unknown, a general practice is to design these methods to fully balance the distribution so that each class has a more or less equal number of samples (Wasikowski, 2009).

We attempted 3 simple and fundamental approaches to resample our dataset: *over-sampling*, *under-sampling*, and a mixture of both (Movin & Jagelid, 2017).

1. *Over-sampling*, or *random over-sampling* (ROS), adds duplicates of randomly selected users from the minority class to the dataset until a preferred ratio is reached.
2. *Under-sampling*, or *random under-sampling* (RUS), randomly eliminates samples from the majority class until preferred ratio has been reached.

3. *Over-under-sampling* is mixture of both ROS and RUS, where the minority class is over-sampled with replacement and majority class is under-sampled without replacement (Movin & Jagelid, 2017).

We use the *ROSE* R-package to implement these resampling approaches.

Experiments with Different Methods

To obtain a good predictor, we tested a variety of different learning methods and compared their performance. The following sections summarize our findings for five methods:

1. Logistic regression,
2. Linear discriminant analysis (LDA),
3. Classification and regression trees (CART),
4. Tree boosting (C5.0), and
5. Stacking.

Justification of Choice of Methods

In our initial trying out of models, support vector machines (SVM), gradient boosting (GBM), and random forests (RF) were also tried, however these were not the best performing on our data according to the *Kappa* score and were dropped from more detailed analysis because of relatively slow computational time (on the order of hours vs. minutes for the other methods). We decided to use the logistic regression and LDA as well-known non-tree classifiers to get a benchmark performance for comparison with the other models.

Decisions trees can be constructed even in the presence of categorical variables, removing the need to create dummy variables, or any encoding of variables. On top of that, they can be interpreted easily. The recursive structure of tree-based models is ideal for uncovering complex dependencies among predictor variables. If some variables are dependent on each other in a nonlinear fashion, tree-based models do a better job at discovering these cases than interaction terms in generalised linear models (Fridley, 2010). There are two common packages for CART, models in R: *tree* and *rpart*.

The *rpart* package is used in this project as one of the tree-based classifiers, also known as recursive partitioning. *Rpart* uses the *Gini index* as the metric for splitting by default, and the cost of adding another split to the tree is assessed with a complexity parameter (cp). It specifies how the cost of a tree is penalized by the number of terminal nodes (Therneau & Atkinson, 2018). In other words, it is used to control the size of the decision tree and to select the optimal tree size. Tree construction will stop if the cost of adding another variable to the decision tree from the current node is above the value of cp. Compared to trees, the output of *rpart* will not show information about deviance at each node, and it may be harder to interpret, with the cost of adding each split to the tree is assessed with cp. Although *rpart* and Tree can produce different results (Fridley, 2010), we picked *rpart* because it is widely considered as a baseline tree-based classifier in the *caret* package.

To combine the power of boosting with a tree-based method, we chose to use *C5.0*. A very powerful technique that uses a combination of decision trees, rules, and boosting (Wu, et al., 2008).

Finally, we chose to try *stacking* to see the effect of combining tree-based and non-tree-based models and because stacking techniques are often the winners of machine learning competitions such as the Kaggle Competition (ML Wave, 2015).

1 Logistic Regression

To evaluate using logistic regression for prediction from our data, we trained five different logistic regression models. First, without rescaling or resampling. Second, with the *Box-Cox transform*. Then, with the *Box-Cox transform* and resampled data for each of three different types of resampling: over-sampling, under-sampling, and both. Test values were computed both by using 10-fold cross-validation built into the *caret* R package on the training data and by explicitly comparing predictions with the test data.

The resampling did not help *sensitivity,* which makes sense because we were trying to predict the majority class. The best model was the one with the *Box-Cox transform* but no resampling. It outperformed all the others in both *Kappa* and s*ensitivity* (true positive rate) scores, although the models based on resampled data had much higher *specificity* (true negative rate) as would be expected thanks to the resampling. Since the *Kappa* score was the main metric we were using to compare models, we concluded that the best logistic regression model was the one trained on data without resampling, but with *Box-Cox transform*. It had a *sensitivity* of 0.9318, *specificity* of 0.5947 and a *Kappa* score was 0.5629. This means it correctly classified 93% of survey results under 50k and 59% of survey results over 50k.

2 Linear Discriminant Analysis (LDA)

Linear discriminant analysis is fairly similar to logistic regression, but it has the advantage that its parameter estimates are more stable than logistic regression when classes are well separated. If there is a small number of predictors and they are approximately normally distributed in each of the classes, LDA is again more stable than the logistic regression model (James, Witten, Hastie, & Tibshirani, 2013).

To evaluate using linear discriminant analysis for prediction on our dataset, we trained five LDA models with the same five input/rescaling combinations as for logistic regression. The best *Kappa* score and *sensitivity* were obtained on the non-resampled dataset and were 0.5269 and 0.9306 respectively, lower than those of logistic regression.

3 Trees: Classification and Regression Trees (CART)

Similar to logistic regression and LDA, we trained the *rpart* tree-based classifier on training data that was not resampled, and on over sampled, under sampled and both over and under sampled training sets. However, we did not use *Box-Cox transformation* to rescale the feature values because tree-based methods by nature are not affected very much if at all by rescaling, unlike numerical-based approaches such as logistic regression and LDA. Tuning was performed automatically by setting an option on the train function of the *caret* package in R and using a tuning grid as with the other methods. The best performing model based on our chosen metrics of *Kappa* and *sensitivity* computed using the test data was the model trained on data without any resampling. It had a *Kappa* score of 0.5812 and a *sensitivity* of 0.9388, better than both logistic regression and LDA.

4 Trees: Boosting C5.0

C5.0 refers to a powerful tree-based machine learning method developed by Australian computer scientist and researcher Ross Quinlan. It uses a combination of decision trees, rules, and boosting to achieve one of the most robust and accurate classifiers (Wu, et al., 2008). C5.0 can also use a winnowing technique to reduce overfitting. Using the *caret* library and a tuning grid, we chose trials = 15 and winnow = FALSE as the best tuning parameters. The number of trials limits boosting and winnowing thins out the set of features used to produce trees which is useful when there are large numbers of features and some of them have little relevance to the classification problem at hand (Rulequest Research, 2017). For this project, there are relatively few features so it seems reasonable that the best tuning parameters say to not use winnowing. There is another C5.0 tuning parameter called *minCases* that controls the depth of the trees, but is not readily adjustable in the *caret* package (Alberg, 2015). Refining the choice of this parameter may have the potential to improve the performance of the C5.0 model even further.

Like with the *rpart* tree-based classifier, *Box-Cox transformation* was not used because rescaling has little effect on tree-based methods. As with logistic regression and *rpart*, the best *Kappa* score and *sensitivity* were obtained for the dataset without resampling. They were 0.6099 and 0.9451 respectively, the best of the models we tried so far.

5 Stacking

Stacking, or model ensembling, is a powerful way to increase model performance by combining several different types of models together and is often the winner of machine learning competitions such as the Kaggle Competition (ML Wave, 2015). When creating an ensemble method, it is good practice to avoid combining highly correlated models. One reason for this is that when models make very similar predictions most of the time there is less benefit in combining them. We checked correlations between the predictions of our best models for logistic, LDA, CART, and C5.0 and logistic and LDA had a correlation of over 0.75. This is not surprising because logistic regression and LDA are fundamentally similar techniques (James, Witten, Hastie, & Tibshirani, 2013). We chose to include LDA in our stacking model because it was less correlated with the other models than logistic regression. To combine the results of the methods within the stacked model, we tried two different methods: logistic regression and classification and regression trees (cart). The stacked model using logistic regression to combine the models performed better in *sensitivity* and AUROC, while the stacked model using cart had a higher *Kappa* score.

Combining the three methods, LDA, CART, and C5.0 using logistic regression, we obtained a classification model with a *Kappa* score of 0.6129, *sensitivity* of 0.9437, and AUROC of 0.920, even better than C5.0 on its own. This was the best model we obtained and is highlighted in yellow in Table 2.

Summary and Interpretation of Results

Table 2 shows a list of the well-performing classifiers we evaluated with the best model in each method, or local optimal model, highlighted with a darker colour hue. All running times are for when the code was run on an Intel(R) Core(TM) i7-7700HQ Lenovo laptop, with 16GB of RAM.

Table 2: Summary of Results and Model Selection

Summary of Results and Model Selection

Classifier	Resampling Approach	Running Time (secs)	Kappa	Sensitivity	Specificity	AUROC
Logistic Regression, with Box Cox transformation	None	23.62	**0.5629**	**0.9318**	**0.5947**	**0.906**
	Over-sampling	30.6	0.5561	0.7999	0.8485	
	Under-sampling	9.76	0.5537	0.7979	0.8491	
	Both	21.43	0.5544	0.7998	0.8463	
Linear Discriminant Analysis, with Box Cox transformation	None	12.48	0.5269	0.9306	0.5559	0.892
	Over-sampling	19.86	0.5128	0.7592	0.8642	
	Under-sampling	6.34	0.5127	0.7569	0.8685	
	Both	12.96	0.5153	0.7596	0.8671	
Classification Trees, using rpart	None	13.75	0.5629	0.9318	0.5947	0.875
	Over-sampling	21.75	0.5128	0.7592	0.8642	
	Under-sampling	5.79	0.5117	0.7562	0.8682	
	Both	12.91	0.5156	0.7601	0.8668	
C5.0	None	223.87	0.6099	0.9451	0.6224	0.917
	Over-sampling	489.61	0.5919	0.8442	0.8095	
	Under-sampling	85.45	0.5889	0.8102	0.8754	
	Both	237.53	0.5919	0.8442	0.8095	
Stacking (rpart, lda, C5.0) with logistic	None	99.67	0.6129	0.9437	0.6289	0.920
Stacking (rpart, lda, C5.0) with cart	None	101.77	0.6246	0.9103	0.7142	0.812

Models trained on non-resampled training sets have generally poor specificity scores because standard classification models (e.g. logistic regression, SVM, decision trees, nearest neighbors) treat all classes as equally important and thus tend to be biased towards the major class in imbalanced problems, producing a high sensitivity score (E. Burnaev, P. Erofeev, & A. Papanov, 2017).

Models trained on resampled training sets produced a lower *Kappa statistic* and *sensitivity*, despite boosting *specificity* favourably, as compared to models trained on non-resampling training sets. Although resampling approaches have improved the prediction of the minority class, it comes with an undesirable trade-off in a drop in *Kappa* and *sensitivity*. This could be due to the nature of random resampling approaches. For over-sampling, the replication of minority class samples likely

led to overfitting. Likewise, for under-sampling, the sample chosen may be biased, resulting in inaccurate results.

Logistic regression (LR) outperforms linear discriminant analysis (LDA) significantly, in terms of the *Kappa statistic*. This could be because LR, unlike LDA, is relatively robust, and it does not require assumptions to be made regarding the distribution of the features, whereas LDA assumes features are normally distributed. Since not all of the features are normally distributed, the usage of LDA is theoretically wrong, as the assumptions are not completely met, despite applying a *Box-cox transformation* (Pohar, Blas, & Turk, 2004).

Tree-based classifier *rpart* displays higher scores compared to LR and LDA because of its flexibility in handling categorical features, and requires minimal assumptions.

The top contender for the best model is a tree-based boosting version of C5.0. Other than being able to detect the relevancy of features, it handles the multi-value features well and mitigates the over-fitting problem by automated pruning. Our final tuned C5.0 a tree-based C5.0 method is accompanied with 15-trial boosting, where 15 separate decision trees or rulesets are combined to make predictions. Figure 16 in Appendix A lists the top 13 features it uses to classify samples from the training set.

We picked the best model by examining 3 main metrics: *Kappa statistic*, *sensitivity*, and Area under ROC curve. The best performing model is the stacked model: *rpart*, LDA, and C5.0 (highlighted in yellow). We combined the predictions of these three individually-tuned models using logistic regression. Its outstanding scores can be credited to its smoothing nature and ability to highlight each base model where it performs best and discredit each base model where it performs poorly (Gorman, 2016).

Conclusion

We obtained our best model through the use of the technique of Stacking, in which we combined three models (LDA, CART, and C5.0), attaining a *sensitivity* of 94.37%. This result means that given the set of feature values (*age*, *education level*, etc.) our model would correctly classify about 94% of people who make less than fifty thousand dollars per year, although it would only correctly classify about 63% of people who make more than that. For a welfare system that should benefit those in greatest need, even at the expense of "wasting" some subsidies on the less deprived, our model could be an effective tool for predicting which people have lower income based only on a set of survey characteristics.

There are some notable take-aways from this project. Having models trained on different resampling methods taught us that resampling a class imbalance dataset might not deliver results compatible with preliminary objectives. Although there is likely no algorithm that performs consistently well every time, understanding the strengths and weakness of each method still gives one an edge over randomly fitting a myriad of models and hoping for the best.

Overall, we are pleased with our approach in optimising predictive performance. Many other algorithms were attempted, but they were computationally slow to train and tune. Given excess time, we may look into implementing parallel processing in R to speed up some of the computationally expensive tasks, like tuning Support Vector Machines and Random Forest models. We could also perform a deeper analysis to justify the grouping of levels in categorical features, because the grouping was done intuitively without thorough justification.

References

Alberg, J. (2015, June 14). *R, caret, and Parameter Tuning C5.0*. Retrieved from Euclidean Technologies: http://www.euclidean.com/machine-learning-in-practice/2015/6/12/r-caret-and-parameter-tuning-c50

E. Burnaev, P. Erofeev, & A. Papanov. (2017, July 12). Influence of Resampling on Accuracy of Imbalanced Classification. *Institute for Information Transmission Problems (Kharkevich Institute) RAS*.

Fridley, J. (2010, February 22). *Tree models in R*. Retrieved from http://plantecology.syr.edu/fridley/bio793/cart.html

Gorman, B. (2016, December 27). *A Kaggler's Guide to Model Stacking in Practice*. Retrieved from http://blog.kaggle.com: http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/

Kohavi, R., & Becker, B. (1996, May 1). *UC Irvine Machine Learning Repository*. Retrieved from https://archive.ics.uci.edu/ml/datasets/Adult

Longadge, R., Dongre, S. S., & Malik, L. (2013, February). Class Imbalance Problem in Data Mining: Review. *International Journal of Computer Science and Network*.

ML Wave. (2015, June 11). *KAGGLE ENSEMBLING GUIDE*. Retrieved from mlwave.com: https://mlwave.com/kaggle-ensembling-guide/

Movin, M., & Jagelid, M. (2017). A Comparison of Resampling Techniques to Handle the Class Imbalance Problem in Machine Learning- Conversion prediction of Spotify Users - A Case Study.

Pearson, R. (2016, 4 12). *The GoodmanKruskal package: Measuring association between categorical variables.* Retrieved from cran.r-project.org: https://cran.r-project.org/web/packages/GoodmanKruskal/vignettes/GoodmanKruskal.html

Pohar, M., Blas, M., & Turk, S. (2004). Comparison of Logistic Regression and Linear Discriminant Analysis: A Simulation Study. *Metodološki zvezki*, pp. 143-161.

Rulequest Research. (2017, March). *C5.0: An Informal Tutorial*. Retrieved from www.rulequest.com: https://www.rulequest.com/see5-unix.html#WINNOWING

Scibilia, B. (2015, March 30). *How Could You Benefit from a Box-Cox Transformation?* Retrieved from The Minitab Blog: http://blog.minitab.com/blog/applying-statistics-in-quality-projects/how-could-you-benefit-from-a-box-cox-transformation

Standard Wisdom LLC. (2011, December 29). *Confusion Matrix – Another Single Value Metric – Kappa Statistic*. Retrieved from http://standardwisdom.com: http://standardwisdom.com/softwarejournal/2011/12/confusion-matrix-another-single-value-metric-kappa-statistic/

Therneau, T. M., & Atkinson, E. J. (2018, February 23). *An Introduction to Recursive Partitioning Using the RPART Routines.* Retrieved from https://cran.r-project.org: https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf

Wasikowski, M. (2009). Combating the Class Imbalance Problem in Small Sample Data Sets.

Wu, X., Kumar, V., Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., . . . Steinberg, D. (2008, January). Top 10 algorithms in data mining. *Knowledge and Information Systems*, pp. 1–37.

Appendix A: Figures

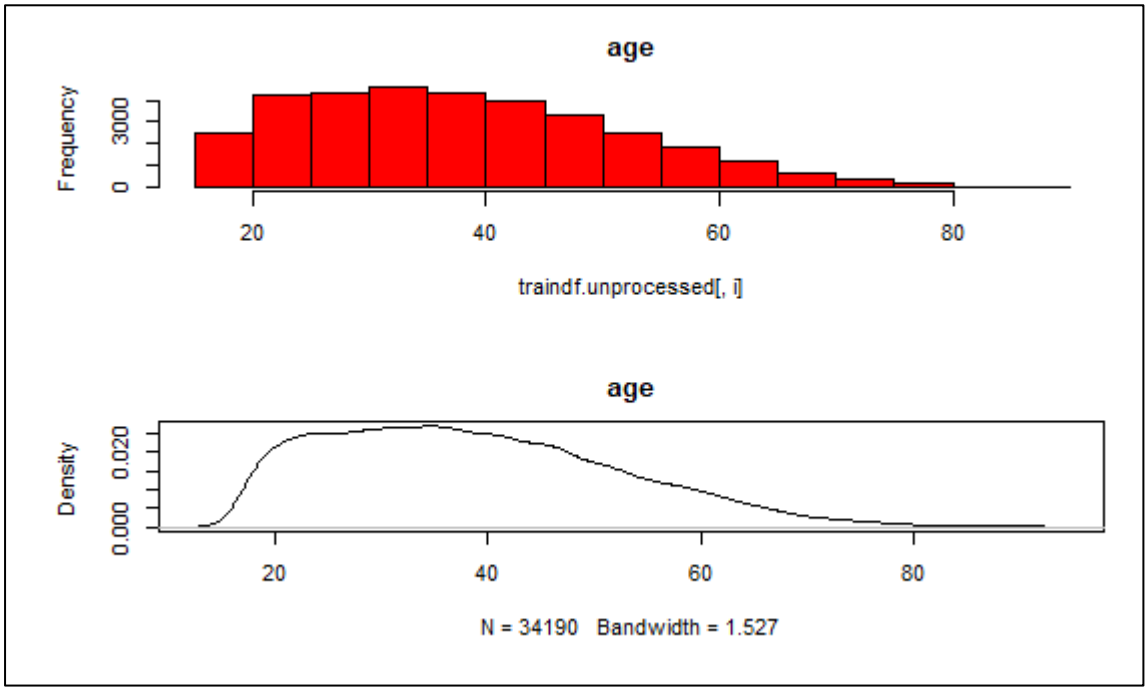

Figure 2: Age Histogram and Kernel Density Plot

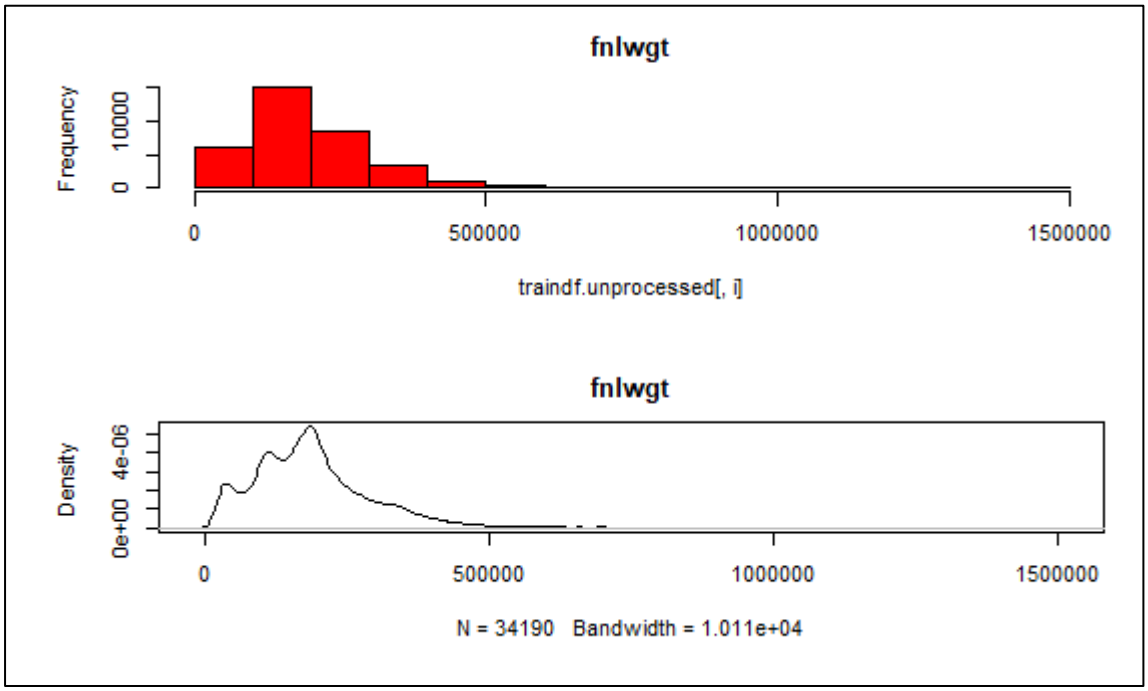

Figure 3: Weight Histogram and Kernel Density Plot

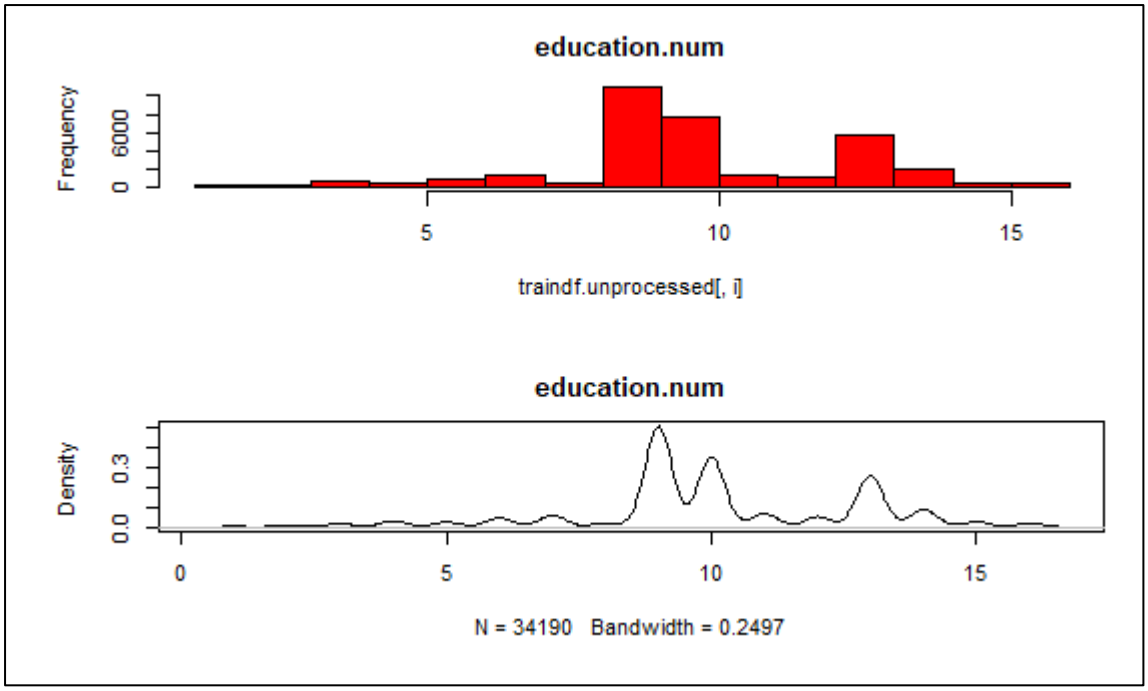

Figure 4: Education Level Histogram and Kernel Density Plot

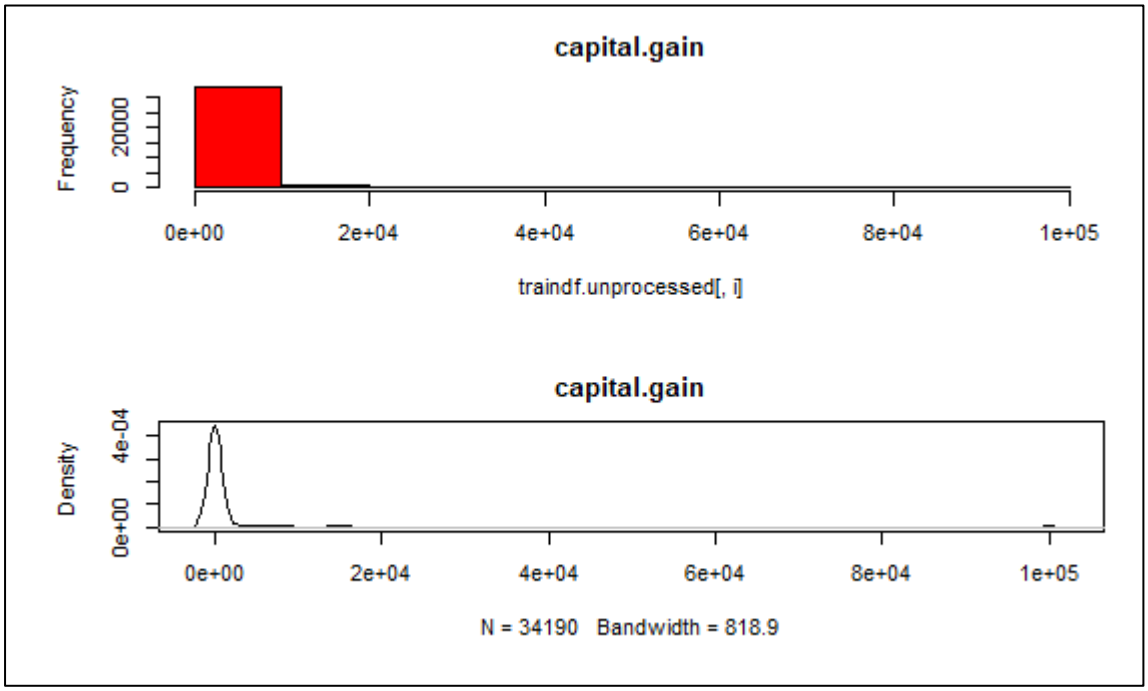

Figure 5: Capital Gain Histogram and Kernel Density Plot

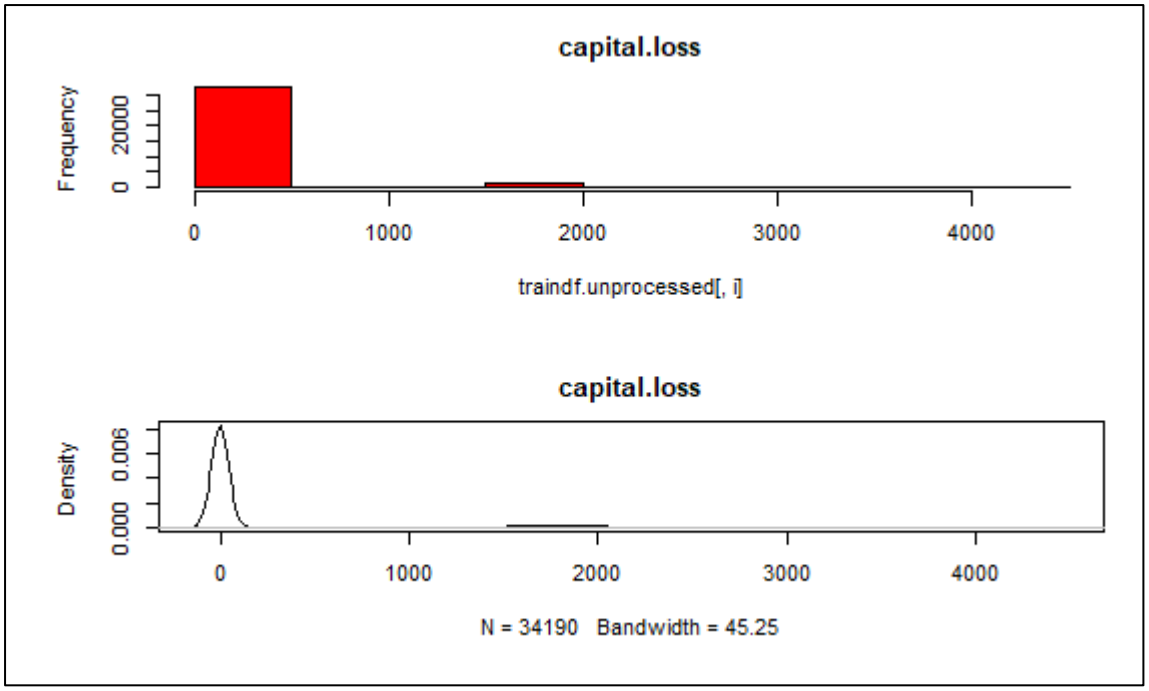

Figure 6: Capital Loss Histogram and Kernel Density Plot

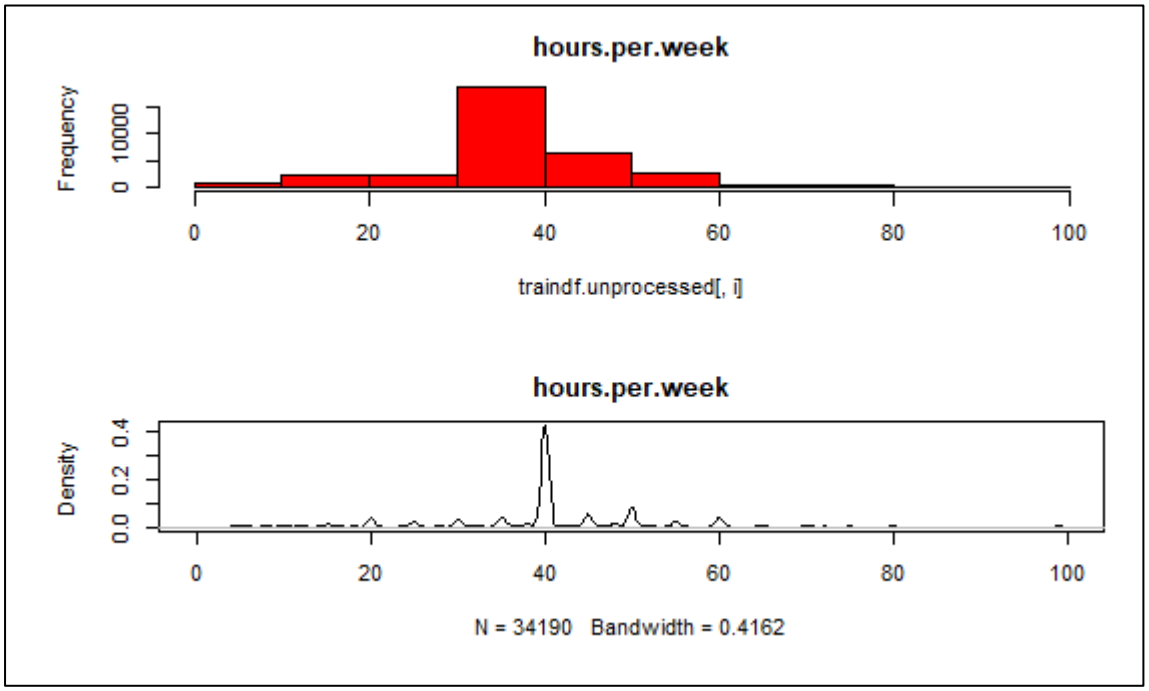

Figure 7: Hours Per Week Histogram and Kernel Density Plot

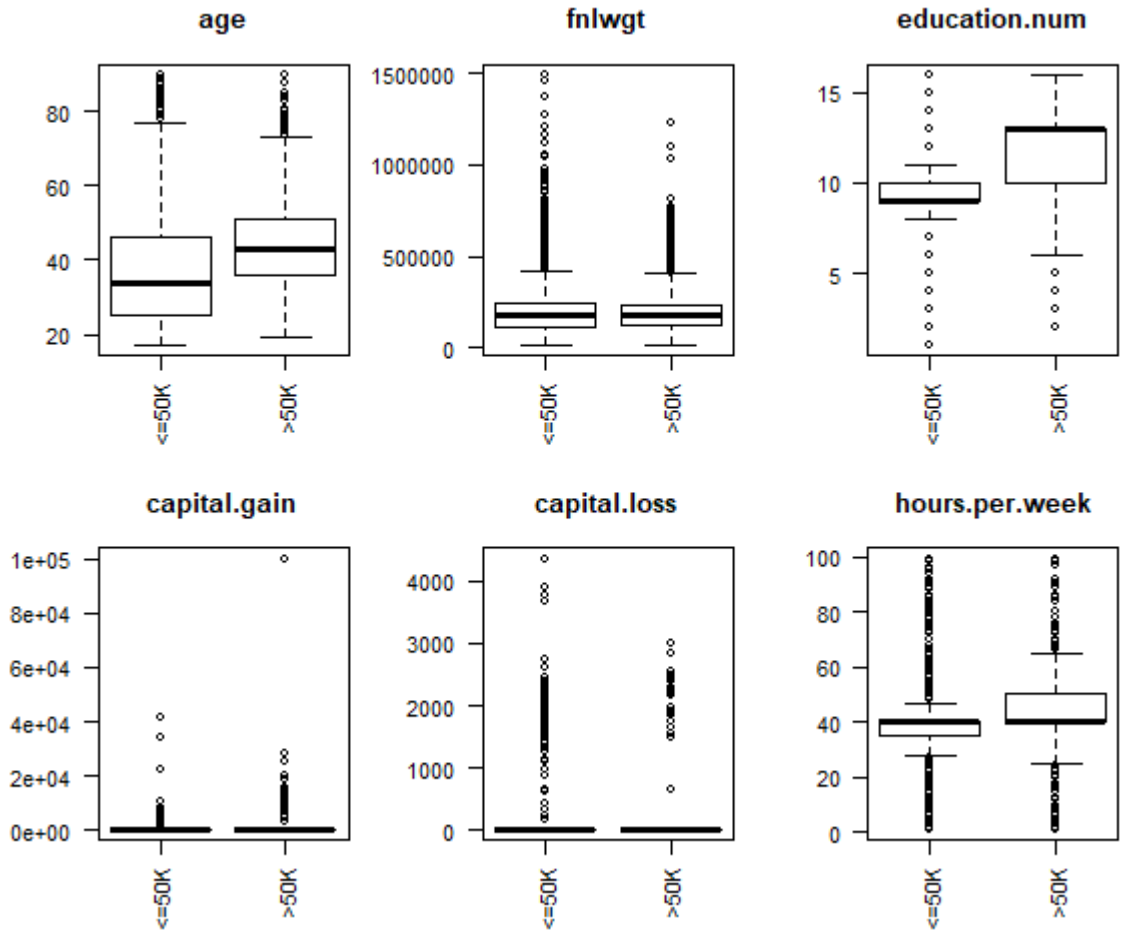

Figure 8: Numeric Feature Boxplots

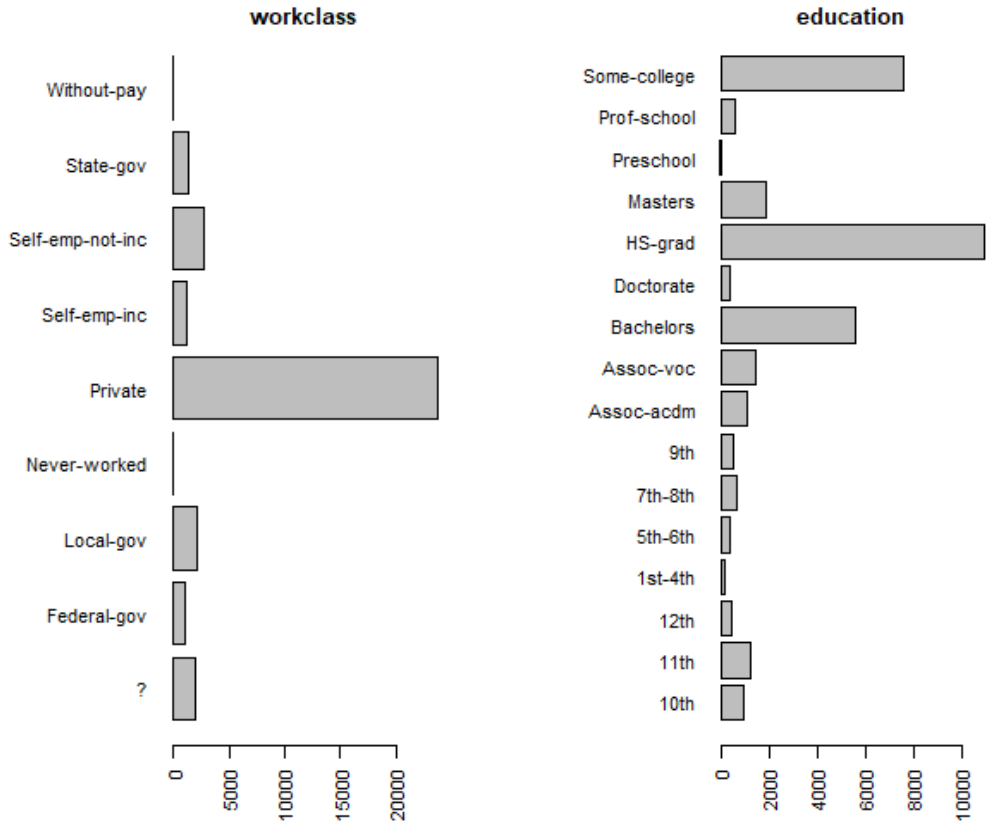

Figure 9: Working Class and Education Bar Plots on Unprocessed Test Data

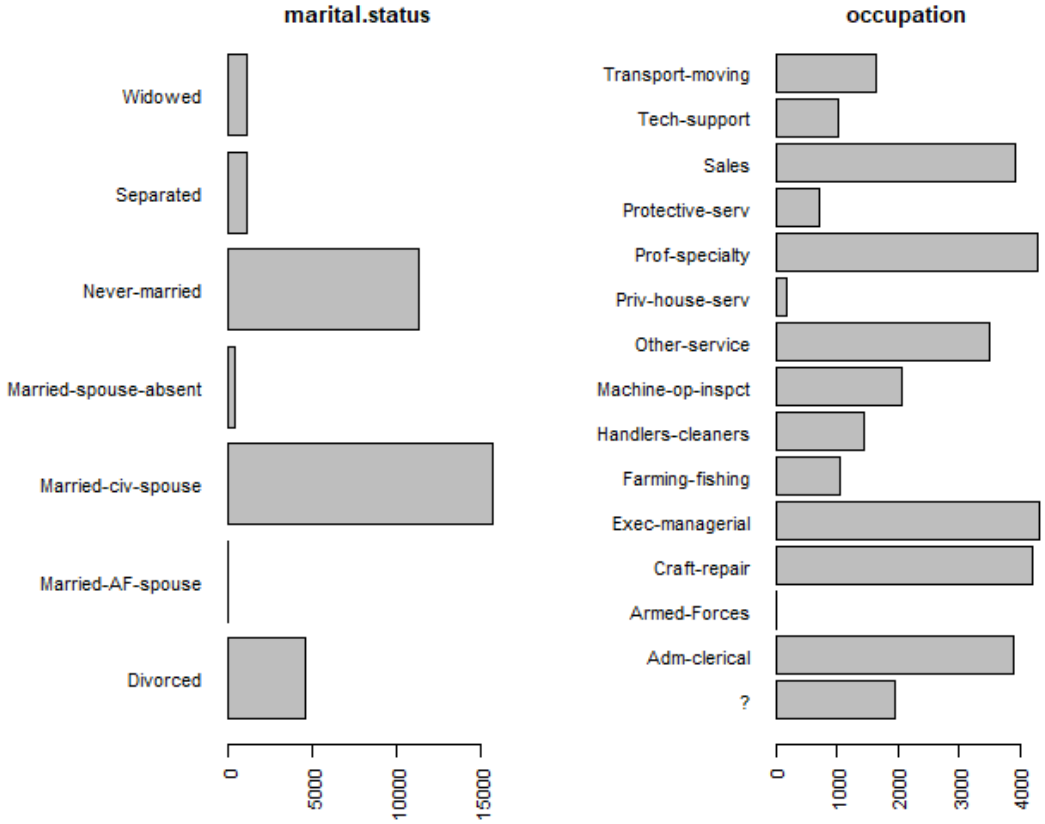

Figure 10: Marital Status and Occupation Bar Plots on Unprocessed Test Data

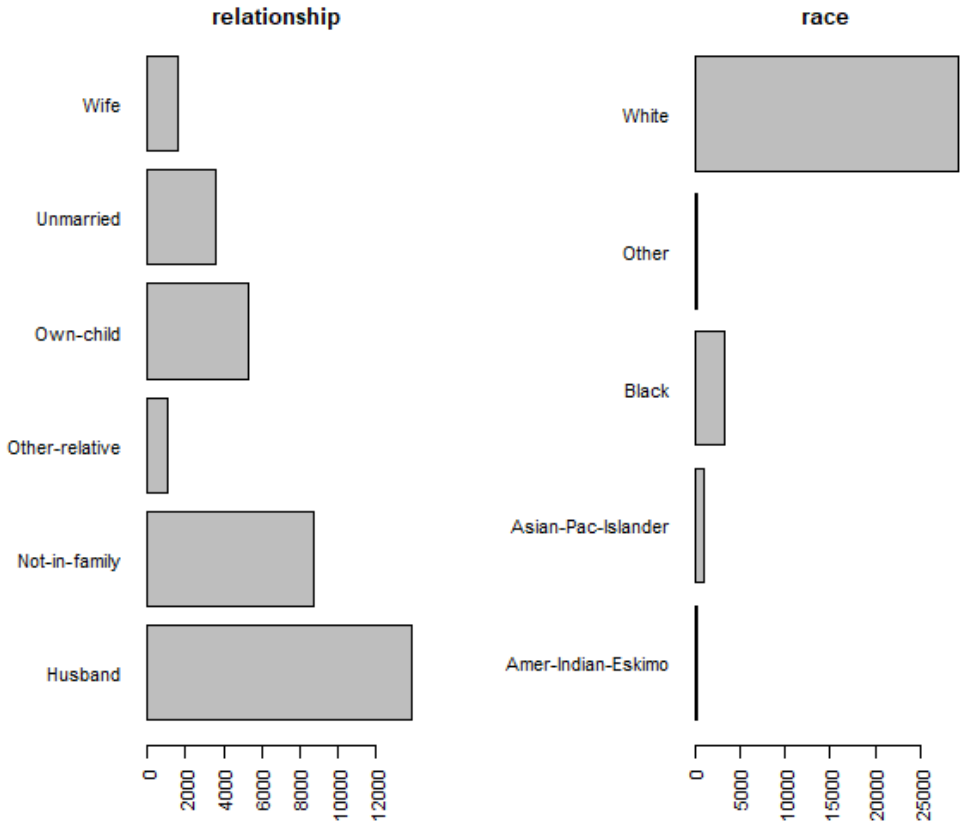

Figure 11: Relationship and Race Bar Plots on Unprocessed Test Data

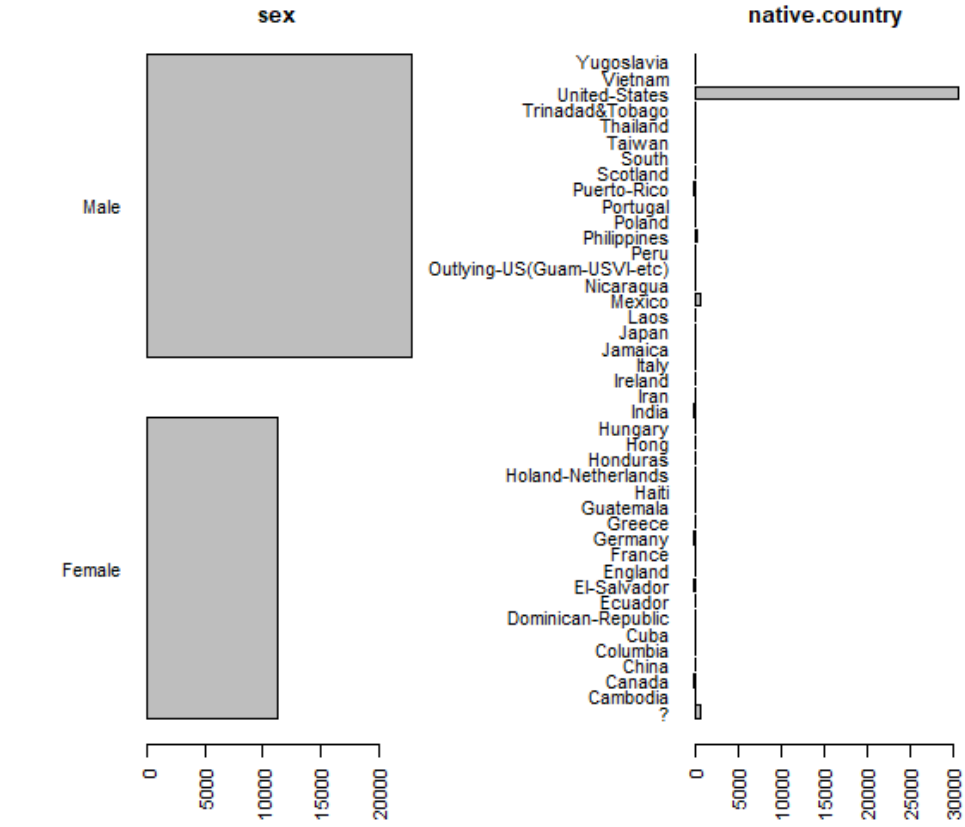

Figure 12: Gender and Country of Origin Bar Plots on Unprocessed Test Data

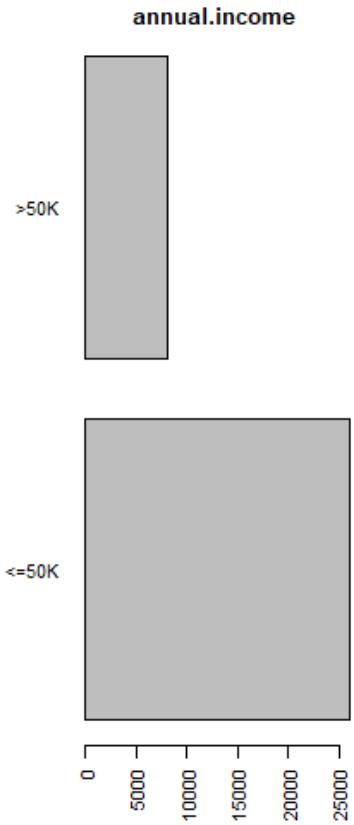

Figure 13: Annual Income Categories Bar Plot on Unprocessed Test Data

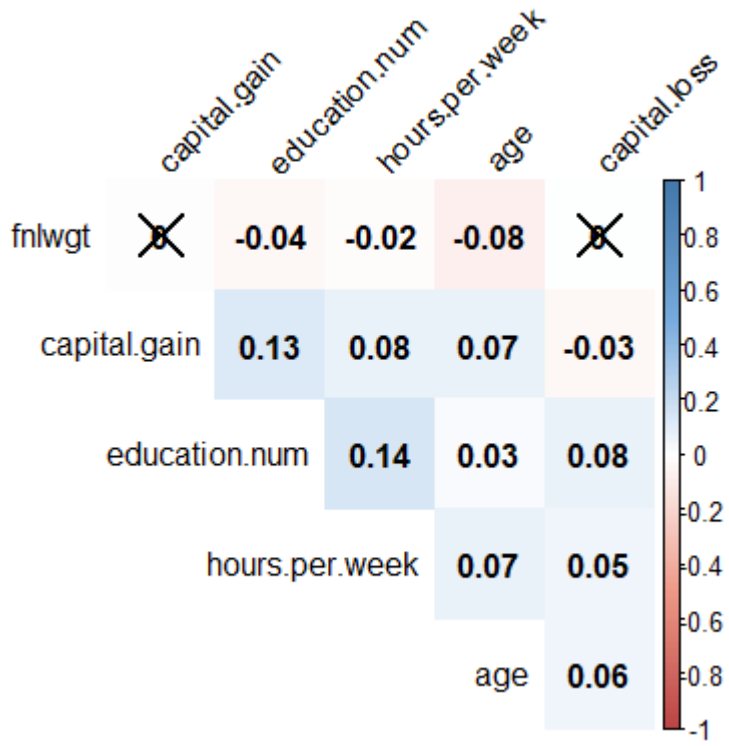

Figure 14: Correlogram showing Correlations between Numeric Features

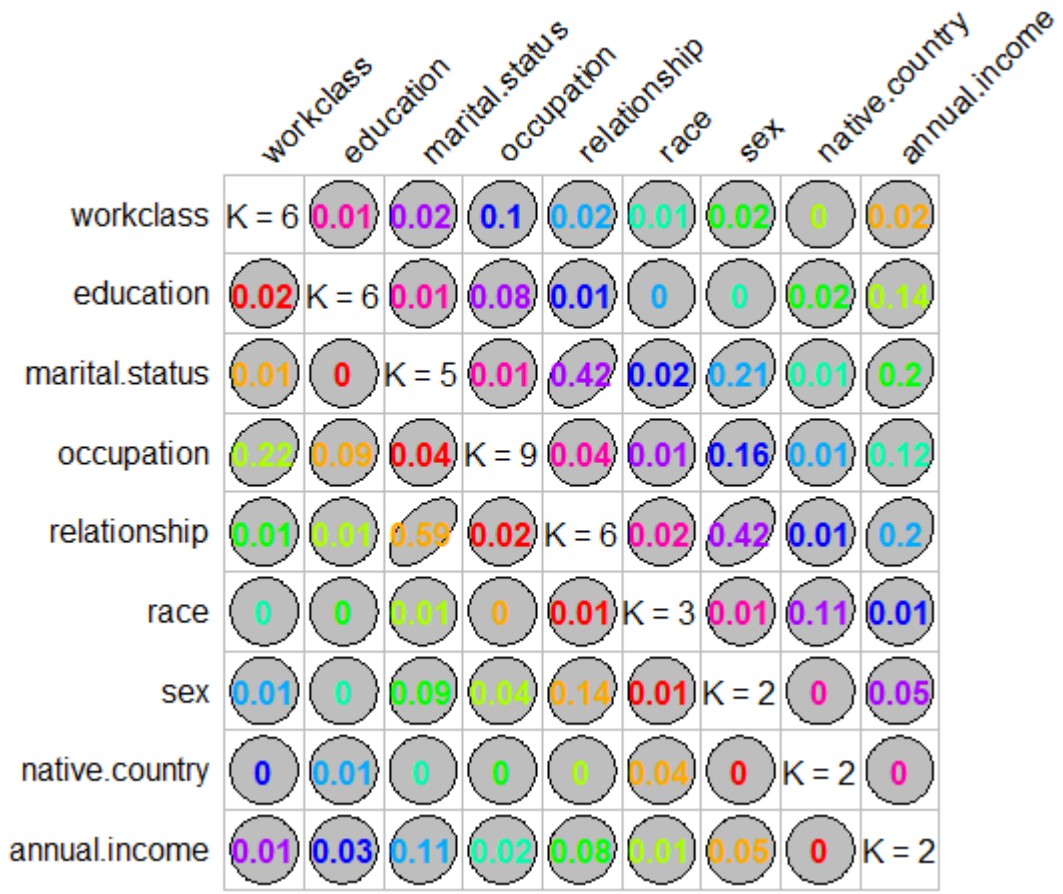

Figure 15: Table of Goodman Kruskal Tau Values for Categorical Feature Relationships

Attribute usage:

100.00%	age
100.00%	educationDropout
100.00%	capital.gain
99.89%	occupationService
99.80%	marital.statusMarried
99.80%	capital.loss
95.99%	relationship Other-relative
95.78%	relationship Own-child
95.69%	occupationBlue-collar
93.22%	educationPhD
88.11%	relationship Unmarried
87.34%	hours.per.week
84.70%	education.num

Figure 16: Top 13 Features used by C5.0 for Classification

Appendix B: R-Code

```
# Dataset Description:
# -> This data set contains weighted census data extracted from the 1994 & 1995 current
population surveys.
# -> The data contains demographic and employment related variables.
# -> Insights? Purpose of predicting more/less than 50k income?

# SOME NOTES:
# -> "# ->" indicate personal comments/intuition
#    "#" indicate the main objective of code

# -> ***Changelog.v4*** Date: 07/03/2018
# -> Updated the arrangement of the code.
# -> Edited section 6.0 & 6.5: Correlation btw numerical features; Correlation btw categorical
features
#    (All correlation analysis now done on training set: traindf)
# -> Removed SVM, GBM, RF (because of computationally slow running time)
# -> Removed OHE (because our best model is a tree-based method C5.0, and it handles
categorical features well)
# -> Added some libraries in Importing Data section 1.0
# -> Added references (Website links) on some sections, to reference where i got the code
from.
#    for sections with no references, i took the code from the R-package documentation.
# -> Added Resampling Approaches section 7.0
# -> Added Visualising Results section 11.0, ROC curve visualisation
# -> Updated running times for some classifiers
# -> Updated Kappa scores
# -> Note: logit is short for logistic regression (i know they are not the same, definition
wise)\

# -> ***Changelog.v5*** Date: 09/03/2018
# -> Updated the arrangement of the code.
# -> Edited Visualising data and Analyse data sections: changed dataset to traindf.unprocessed
# -> Added tree diagram for rpart for visualisation
# -> Added final results summary of C5.0 code for intepretation
# -> Added comments in Resampling Approaches section


# -> ***Changelog.v6*** Date: 11/03/2018
# -> Updated running times for each model


# 1.0 Importing Data
================================================================================================
setwd("C:/Users/longwind48/Google Drive/Programming/MA429/Mock Project")

# Extracting column names from adult.names file
ci_names_S <- scan("data/adult.names", skip = 96, sep = ':', what = list(''))
ci_names_S <- unlist(ci_names_S)
ci_names_S <- ci_names_S[c(TRUE, FALSE)]
ci_names_S <- c(ci_names_S, 'annual-income')
summary(ci_names_S)

# Import .data file with column names
cis_df <- read.table("data/adult.data", header = FALSE, sep=",",
                     col.names = ci_names_S)

# Import .test file with column names
cis_df_test <- read.table("data/adult.test", skip = 1, header = FALSE, sep=",",
                     col.names = ci_names_S)
# -> Turns out the response variable has a dot on the levels
# -> Let's remove it.
levels(cis_df_test$annual.income)[levels(cis_df_test$annual.income)==" <=50K."] <- " <=50K"
levels(cis_df_test$annual.income)[levels(cis_df_test$annual.income)==" >50K."] <- " >50K"

# Find the proportion of split, for spliting it back later
prop_split <- as.numeric(dim(cis_df)[1]/(dim(cis_df)[1]+dim(cis_df_test)[1]))

# Merge 2 datasets
dataset <- rbind(cis_df, cis_df_test)
# -> We merge 2 datasets because it could possibly improve our pre-modelling analysis due to
the increase in data points.

# Load the libraries
```

```
library(plyr); library(dplyr); library(psych); library(caret); library(car); library(C50);
library(e1071);
library(caTools); library(ggplot2)
library(randomForest); library(ROCR); library(MASS)
library(rpart); library(MASS); library(GoodmanKruskal); library(corrplot);
library(caretEnsemble)
#devtools::install_github("collectivemedia/tictoc")
#devtools::install_github("easyGgplot2", "kassambara")
library(tictoc); library(easyGgplot2); library(ggthemes); library(PRROC); library(pROC);
library(randomcoloR)

# 1.5 Spliting the merged dataset
================================================================================
# -> Data is split in 2 different sets: training and testing sets

# Shrink the dataset to train our models faster, 10%
# cutoff = round(0.1*nrow(dataset))
# dataset <- dataset[1:cutoff,]
# create a list of 70% of the rows in the original dataset we can use for training
set.seed(7)
test.index <- createDataPartition(dataset$annual.income, p=0.7, list=FALSE)
# select 30% of the data for testing
testdf.unprocessed <- dataset[-test.index,]
# use the remaining 70% of data for creating training dataset
traindf.unprocessed <- dataset[test.index,]


# 2.0 Analyse Data
================================================================================================

#View(traindf.unprocessed)
head(traindf.unprocessed)
dim(traindf.unprocessed)
# -> We have 34190 instances to work with and 15 features/attributes.
str(traindf.unprocessed)
# -> We want to predict the last feature (annual.income)
# -> This is a binary classification problem, since there are only 2 levels for our reponse
variable.

# Look at how balanced the classes in feature (annual.income) are
percentage_cis <- prop.table(table(traindf.unprocessed$annual.income)) * 100
cbind(freq=table(traindf.unprocessed$annual.income), percentage=percentage_cis)
# -> We can see that 76% of instances have <=50k annual income.
# -> This is an indicator of a class imbalance problem.
# -> Different metrics should be used. Kappa?

# Looking at what type of features do we have to work with
sapply(traindf.unprocessed, class)
# -> 6 features are numerical, 8 features are categorical

sapply(traindf.unprocessed, levels)
# -> features like native.country has 42 levels, makes more sense to reduce it because
'United-States' significantly
#    outweighs the rest of the levels
# -> We could also combine some of the low frequency levels in some catergorical features.
WHY? i forgot the reason...
# -> Some of the levels in our categorical features are similar,
#    like 'seperated' and 'divorced', 'local-gov' and 'state-gov', so many more.

# Let's take a look at the counts of each level in our categorical feature.
summary(traindf.unprocessed, maxsum = 42)
# -> There are many levels with low frequency, this gives us an idea of what to combine and
what not to combine.
# -> We can also take a look at the histograms.
# -> But i already look at them, and made the call to combine levels in some of the
categorical features, under section 4


# 3.0 Visualize traindf.unprocessed
================================================================================================

# Discover whether a gender income gap exist
ggplot(traindf.unprocessed, aes(annual.income, fill = sex)) +
    geom_bar(stat = "count", position = 'dodge')+
    theme_few() +
    xlab("annual.income") +
    ylab("count") +
```

```
    scale_fill_discrete(name = "sex") +
    ggtitle("Gender and Income")

# Discover whether a racial income gap exist
ggplot(traindf.unprocessed, aes(annual.income, fill = race)) +
    geom_bar(stat = "count", position = 'dodge')+
    theme_few() +
    xlab("annual.income") +
    ylab("Count") +
    scale_fill_discrete(name = "race") +
    ggtitle("Race and Income")

# Shows the proportion of groups in the features earning more than 50k
prop.table(table(traindf.unprocessed$annual.income[traindf.unprocessed$race==' Black']))
prop.table(table(traindf.unprocessed$annual.income[traindf.unprocessed$race==' White']))
prop.table(table(traindf.unprocessed$annual.income[traindf.unprocessed$race==' Other']))
prop.table(table(traindf.unprocessed$annual.income[traindf.unprocessed$sex==' Male']))
prop.table(table(traindf.unprocessed$annual.income[traindf.unprocessed$sex==' Female']))

# 3.1 Create barplots for each categorical feature
par(mfrow=c(2,2))
par(las=2)
for (i in 1:length(traindf.unprocessed)) {
    if (class(traindf.unprocessed[,i])=="factor") {
        plot(traindf.unprocessed[,i], main=colnames(traindf.unprocessed)[i], cex.names=0.65,
horiz=TRUE)
    }
}
# -> (workclass) There are levels which are low in frequency; i.e. Without-pay and Never-
worked
#    Can possibly combine state-gov, local-gov, federal-gov
#    also, self-emp-not-inc and self-emp-inc.
# -> (education) There are levels which can be combined too, i.e. all the grades.
# -> (marital-status) Married-AF-spouse is basically non-existent
# -> (occupation) probably can group some low freq occupations into 'others'
# -> (relationship) possibly linked to marital-status
#    Can combine husband and wife to form a new category 'married'
# -> (race) can combine low frequency races to 'others'
# -> (sex) it is clear that there are more males than females.
# -> (native-country) most of the people are us citizens,
#    probably not helpful if we were to consider all low frequency categories,
#    Could keep mexico, or change into continents.

# 3.2 Create boxplots for each numerical feature
par(mfrow=c(2,3))
par(cex=0.7, mai=c(0.3,0.8,0.5,1))
for (i in 1:length(traindf.unprocessed)) {
    if (class(traindf.unprocessed[,i])=="integer" | class(traindf.unprocessed[,i])=="numeric"
) {
        boxplot(traindf.unprocessed[,i]~traindf.unprocessed[,15],
main=colnames(traindf.unprocessed)[i])
    }
}
# -> Should we deal with outliers?

# 3.3 Create Histograms and Kernel density plots for each numerical feature
par(mfrow=c(2,1))
par(cex=0.7, mai=c(0.3,0.8,0.5,1))
for (i in 1:length(traindf.unprocessed)) {
    if (class(traindf.unprocessed[,i])=="integer" | class(traindf.unprocessed[,i])=="numeric"
) {
        hist(traindf.unprocessed[,i], breaks=12, col="red",
main=colnames(traindf.unprocessed)[i])
        plot(density(traindf.unprocessed[,i]), main=colnames(traindf.unprocessed)[i])
    }
}
# -> (age) There is a clear right skew, should probably try to make it normal, or not?

# 3.4 Visualising the relationship between each pair for numerical features
pairs.panels(traindf.unprocessed[,c(1,3,5,11:13)], gap=0, bg=c('red',
'blue')[traindf.unprocessed$annual.income], pch=2)

# Use qqplot to see how well 'age' variable fits a normal distribution
qqnorm(traindf.unprocessed$age); qqline(traindf.unprocessed$age)
# -> Visual inspection is usually unreliable,
#    possible to use a significance test comparing the sample distribution to a normal one i
library(nortest)
```

```
ad.test(traindf.unprocessed$age)
cvm.test(traindf.unprocessed$age)
# -> For both Cramer-von Mises test for normality and Anderson-Darling test for normality, p
values are <0.05,
#    This most likely imply that it is not normally distributed.
# -> Should we do a power transform to make it more normal?


# Understanding capital.gain and capital.loss
par(mfrow=c(1,2))
par(cex=0.7, mai=c(0.3,0.8,0.5,1))
boxplot(traindf.unprocessed$capital.gain[traindf.unprocessed$capital.gain>0]~traindf.unprocess
ed$annual.income[traindf.unprocessed$capital.gain>0])
# ->
library(Hmisc)
describeBy(traindf.unprocessed$capital.gain, group = traindf.unprocessed$annual.income, mat =
TRUE )
summary(traindf.unprocessed$annual.income)
summary(traindf.unprocessed$annual.income[traindf.unprocessed$capital.gain>0])
summary(traindf.unprocessed$annual.income[traindf.unprocessed$capital.gain==0])
describe(traindf.unprocessed$annual.income[traindf.unprocessed$capital.loss>0])
describe(traindf.unprocessed$annual.income[traindf.unprocessed$capital.loss==0])
# -> Looks like capital gain and captial loss are not reliable indicators of having an income
>50k
# -> Out of 25% of instances with >50k, 6k has no capital investments 1.6k has cap gain



# 4.0 Cleaning Data
================================================================================================
# -> Cleaning the merged dataset to split it again later.
# -> After our univariate and multivariate analysis, we proceed to clean the merged dataset

# 4.1 Dealing with (native.country) feature
# Merge all levels with low frequency into one level 'Others'
others <- levels(dataset[,14])[-40]
dataset[,14] <- recode(dataset[,14], "c(others)='Others'")

# 4.2 Dealing with (race) feature
others_n <- levels(dataset$race)[1:2]
dataset$race <- recode(dataset$race, "c(others_n)=' Other'")

# 4.3 Dealing with (education) feature
others_e <- levels(dataset$education)[c(1:7,14)]
dataset$education <- recode(dataset$education, "c(others_e)='Dropout'")
others_e2 <- levels(dataset$education)[c(1:2)]
dataset$education <- recode(dataset$education, "c(others_e2)='Associates'")
others_e3 <- levels(dataset$education)[c(3,6)]
dataset$education <- recode(dataset$education, "c(others_e3)='HSGrad'")
others_e3 <- levels(dataset$education)[c(2,4)]
dataset$education <- recode(dataset$education, "c(others_e3)='PhD'")

# 4.4 Dealing with (workclass) feature
# -> “Never worked” and “Without-Pay” are small groups, we can combine them into 'Not-working'
level
others_wc <- levels(dataset$workclass)[c(6,7)]
dataset$workclass <- recode(dataset$workclass, "c(others_wc)='Self-employed'")
others_wc1 <- levels(dataset$workclass)[c(4,7)]
dataset$workclass <- recode(dataset$workclass, "c(others_wc1)='Not-working'")
others_wc2 <- levels(dataset$workclass)[c(3,5)]
dataset$workclass <- recode(dataset$workclass, "c(others_wc2)='Non-federal-gov'")

# 4.5 Dealing with (occupation) feature
others_o <- levels(dataset$occupation)[c(4,6,7,8,15)]
dataset$occupation <- recode(dataset$occupation, "c(others_o)='Blue-collar'")
others_o1 <- levels(dataset$occupation)[c(5,6)]
dataset$occupation <- recode(dataset$occupation, "c(others_o1)='Service'")
others_o2 <- levels(dataset$occupation)[c(6,8)]
dataset$occupation <- recode(dataset$occupation, "c(others_o2)='Other-occ'")

# 4.6 Dealing with (marital.status) feature
others_ms <- levels(dataset$marital.status)[c(4,6)]
dataset$marital.status <- recode(dataset$marital.status, "c(others_ms)='Not-w-spouse'")
others_ms1 <- levels(dataset$marital.status)[c(2,3)]
dataset$marital.status <- recode(dataset$marital.status, "c(others_ms1)='Married'")

# 4.7 Dealing with (annual.income) feature
```

```r
levels(dataset$annual.income) <- c("less", "more")

# -> Now, the there are lesser levels in some of the categorical features,
#    we can then proceed to analysis.

# 5.0 Spliting the merged dataset
================================================================================
# -> Data is split in 2 different sets: training and testing sets

# Shrink the dataset to train our models faster, 10%
# cutoff = round(0.1*nrow(dataset))
# dataset <- dataset[1:cutoff,]
# Use the same index we used to split earlier
test.index
# Select 30% of the data for testing
testdf <- dataset[-test.index,]
# Use the remaining 70% of data for creating training dataset
traindf <- dataset[test.index,]

# 6.0 Correlations between categorical variables
================================================================

# -> Relationship and marital-status sounds correlated just by the names, let's check using a
chisquared test
table(traindf$marital.status, traindf$relationship)
chisq.test(traindf$marital.status, traindf$relationship, correct=FALSE)
# -> A chi-squared value indicates a substantial relationship between two variables.
# -> We can reject the null hypothesis since our p-value is smaller than 0.05
#    and conclude that our variables are likely to have a significant relationship.

# Check for relationships between all categorical variables
subset <- c(2,4,6:10,14,15)
GKmatrix <- GKtauDataframe(traindf[,subset])
plot(GKmatrix)
# -> Tau value from relationship to marital.status is 0.59, indicating a strong association
# -> Good to see that among all categorical variables, only maritalstatus and rship have
strong associations.
GKtau(traindf$marital.status, traindf$relationship)
# -> The Goodman-Kruskal tau measure: knowledge of marital.status is predictive of
relationship, and similar otherwise.
# -> Reference: https://cran.r-
project.org/web/packages/GoodmanKruskal/vignettes/GoodmanKruskal.html

# 6.5 Correlations between numerical variables
================================================================

# -> To show this, we combining correlogram with the significance test,
# -> Reference: http://www.sthda.com/english/wiki/visualize-correlation-matrix-using-
correlogram
# Build a function to compute a matrix of p-values
cor.mtest <- function(mat, ...) {
    mat <- as.matrix(mat)
    n <- ncol(mat)
    p.mat<- matrix(NA, n, n)
    diag(p.mat) <- 0
    for (i in 1:(n - 1)) {
        for (j in (i + 1):n) {
            tmp <- cor.test(mat[, i], mat[, j], ...)
            p.mat[i, j] <- p.mat[j, i] <- tmp$p.value
        }
    }
    colnames(p.mat) <- rownames(p.mat) <- colnames(mat)
    p.mat
}
cor <- cor(traindf[,c(1,3,5,11,12,13)])
p.mat <- cor.mtest(traindf[,c(1,3,5,11,12,13)])

# Build a correlogram
col <- colorRampPalette(c("#BB4444", "#EE9988", "#FFFFFF", "#77AADD", "#4477AA"))
corrplot(cor, method="color", col=col(200),
         type="upper", order="hclust",
         addCoef.col = "black", # Add coefficient of correlation
         tl.col="black", tl.srt=45, #Text label color and rotation
         # Combine with significance
         p.mat = p.mat, sig.level = 0.01,
         # hide correlation coefficient on the principal diagonal
         diag=FALSE)
```

```r
# -> correlations with p-value > 0.05 are considered as insignificant, and crosses are added
for those.
# -> Don't seem to have any correlation among numerical variables, which is a good thing.


# 7.0 Resampling Approaches
================================================================================

table(traindf$annual.income)

percentage_cis_resampled <- prop.table(table(traindf.resampled$annual.income)) * 100
cbind(freq=table(traindf.resampled$annual.income), percentage=percentage_cis_resampled)

library(ROSE)
# Over-sampling
traindf.resampled.over <- ovun.sample(annual.income ~ ., data = traindf, method = "over",N =
52018, seed=7)$data
table(traindf.resampled.over$annual.income)

# Under-sampling
traindf.resampled.under <- ovun.sample(annual.income ~ ., data = traindf, method = "under",N =
16362, seed=7)$data
table(traindf.resampled.under$annual.income)

# Over-Under-Sampling
# -> Minority class is oversampled with replacement and majority class is undersampled without
replacement.
traindf.resampled.both <- ovun.sample(annual.income ~ ., data = traindf, method = "both",
p=0.5, N = 34190, seed=7)$data
table(traindf.resampled.both$annual.income)


# 8.0 Evaluate Algorithms ====================================================
library(caret)
stats <- function (data, lev = NULL, model = NULL)  {
    c(postResample(data[, "pred"], data[, "obs"]),
      Sens = sensitivity(data[, "pred"], data[, "obs"]),
      Spec = specificity(data[, "pred"], data[, "obs"]))
}

control <- trainControl(method="cv", number=10, summaryFunction = stats, classProbs = TRUE)
# -> The function trainControl can be used to specifiy the type of resampling,
#    in this case, 10-fold cross validation.
metric <- "Kappa"
# control1 <- trainControl(method="repeatedcv", number=10, repeats=3, search="random")
# -> Kappa or Cohen's Kappa is like classification accuracy,
#    except that it is normalized at the baseline of random chance on your dataset.
# -> A more useful measure to use on problems that have an imbalance in the classes

# -> We know we have some skewed distributions,
#    We can use a type of power transform to adjust and normalize these distributions.
# -> A good choice will be Box-cox transform, by using caret's Preprocess parameter.
# -> Some algorithms, like tree-based algorithms are usually invariant to transforms.
# -> We will attempt to fit our model in the following combinations:
#    (Unenconded features/OHE features, no transformation/normalisation/Box-Cox transform)
# -> At each model, before we try a different classification algorithm,
#    we will pick the best combination and attempt to tune to optimise the CV Kappa score.


# LOGIT ********************************************************************************
set.seed(7)
fit.logit <- train(annual.income~.-fnlwgt, data=traindf, method="glm", family="binomial",
                   metric=metric, trControl=control)

set.seed(7)
fit.logit5 <- train(annual.income~.-fnlwgt, data=traindf, method="glm", family="binomial",
                    preProcess = c('BoxCox'),
                    metric=metric, trControl=control)
# -> fit.logit5: 23.62 sec elapsed

set.seed(7)
fit.logit6 <- train(annual.income~.-fnlwgt, data=traindf.resampled.both, method="glm",
family="binomial",
                    preProcess = c('BoxCox'),
                    metric=metric, trControl=control)
# -> fit.logit6: 21.43 sec elapsed
set.seed(7)
```

```
fit.logit7 <- train(annual.income~.-fnlwgt, data=traindf.resampled.over, method="glm",
family="binomial",
                    preProcess = c('BoxCox'),
                    metric=metric, trControl=control)
# -> fit.logit7: 30.6 sec elapsed

set.seed(7)
fit.logit8 <- train(annual.income~.-fnlwgt, data=traindf.resampled.under, method="glm",
family="binomial",
                    preProcess = c('BoxCox'),
                    metric=metric, trControl=control)
# -> fit.logit8: 9.76 sec elapsed


results.logit <- resamples(list(logit=fit.logit, logit5=fit.logit5, logit6=fit.logit6,
logit7=fit.logit7, logit8=fit.logit8))
summary(results.logit)
dotplot(results.logit, metric='Kappa')

# Get true test kappa score
predictions.logit5 <- predict(fit.logit5, newdata = testdf)
predictions.logit5.prob <- predict(fit.logit5, newdata = testdf, type='prob')$less
predictions.logit6 <- predict(fit.logit6, newdata = testdf)
predictions.logit6.prob <- predict(fit.logit6, newdata = testdf, type='prob')$less
roc(testdf$annual.income, predictions.logit6.prob)
predictions.logit7 <- predict(fit.logit7, newdata = testdf)
predictions.logit7.prob <- predict(fit.logit7, newdata = testdf, type='prob')$less
roc(testdf$annual.income, predictions.logit7.prob)

predictions.logit8 <- predict(fit.logit8, newdata = testdf)
confusionMatrix(predictions.logit5, testdf$annual.income)
confusionMatrix(predictions.logit6, testdf$annual.income)
confusionMatrix(predictions.logit7, testdf$annual.income)
confusionMatrix(predictions.logit8, testdf$annual.income)

# -> OHE = Unencoded
# -> Original > Resampled
# -> (Fitted on oversampled train set: fit.logit7) Kappa : 0.5561, Sensitivity : 0.7999,
Specificity : 0.8485
# -> (Fitted on undersampled train set: fit.logit8) Kappa : 0.5537, Sensitivity : 0.7979,
Specificity : 0.8491
# -> (Fitted on both over&undersampled train set: fit.logit8) Kappa : 0.5544, Sensitivity :
0.7998, Specificity : 0.8463
# -> (Fitted on original train set: fit.logit5) Kappa : 0.5629, Sensitivity : 0.9318,
Specificity : 0.5947


# LDA
*********************************************************************************************

set.seed(7)
fit.lda <- train(annual.income~.-fnlwgt, data=traindf, method='lda', preProcess = c('scale',
'center'),
                 metric=metric, trControl=control)
# -> fit.lda: 5.64 sec elapsed

set.seed(7)
fit.lda1 <- train(annual.income~.-fnlwgt, data=traindf, method='lda', preProcess = 'BoxCox',
                  metric=metric, trControl=control)
# -> fit.lda1: 12.48 sec elapsed

set.seed(7)
fit.lda2 <- train(annual.income~.-fnlwgt, data=traindf.resampled.over, method='lda',
preProcess = 'BoxCox',
                  metric=metric, trControl=control)
# -> fit.lda2: 19.86 sec elapsed

set.seed(7)
fit.lda3 <- train(annual.income~.-fnlwgt, data=traindf.resampled.under, method='lda',
preProcess = 'BoxCox',
                  metric=metric, trControl=control)
# -> fit.lda3: 6.34 sec elapsed

set.seed(7)
fit.lda4 <- train(annual.income~.-fnlwgt, data=traindf.resampled.both, method='lda',
preProcess = 'BoxCox',
                  metric=metric, trControl=control)
```

```
# -> fit.lda4: 12.96 sec elapsed

results.lda <- resamples(list(lda=fit.lda, lda1=fit.lda1, lda2=fit.lda2, lda3=fit.lda3,
lda4=fit.lda4))
summary(results.lda)
dotplot(results.lda, metric=metric)

# Get true test kappa score
predictions.lda <- predict(fit.lda, newdata = testdf)
predictions.lda1 <- predict(fit.lda1, newdata = testdf)
predictions.lda1.prob <- predict(fit.lda1, newdata = testdf, type='prob')$less
predictions.lda2 <- predict(fit.lda2, newdata = testdf)
predictions.lda3 <- predict(fit.lda3, newdata = testdf)
predictions.lda4 <- predict(fit.lda4, newdata = testdf)
confusionMatrix(predictions.lda, testdf$annual.income)
confusionMatrix(predictions.lda1, testdf$annual.income)
confusionMatrix(predictions.lda2, testdf$annual.income)
confusionMatrix(predictions.lda3, testdf$annual.income)
confusionMatrix(predictions.lda4, testdf$annual.income)
# -> OHE = Uncoded
# -> (Fitted on oversampled train set: fit.lda2) Kappa : 0.5128, Sensitivity : 0.7592,
Specificity : 0.8642
# -> (Fitted on undersampled train set: fit.lda3) Kappa : 0.5127, Sensitivity : 0.7569,
Specificity : 0.8685
# -> (Fitted on both over&undersampled train set: fit.lda4) Kappa : 0.5153, Sensitivity :
0.7596, Specificity : 0.8671
# -> (Fitted on original train set: fit.lda1) Kappa : 0.5269, Sensitivity : 0.9306,
Specificity : 0.5559
# -> LOGIT(0.5629) > LDA(0.5269)


# TREES
**********************************************************************************************
**
# TREES: Cart
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
# -> Short for Classification and Regression Trees

tic('fit.cart')
set.seed(7)
fit.cart <- train(annual.income~.-fnlwgt, data=traindf, method="rpart",
                  parms = list(split = "information"), #or 'information'
                  metric=metric, trControl=control, tuneLength = 10)
toc()
# -> fit.cart: 13.75 sec elapsed


tic('fit.cart1')
set.seed(7)
fit.cart1 <- train(annual.income~.-fnlwgt, data=traindf.resampled.over, method="rpart",
                  parms = list(split = "information"), #or 'information'
                  metric=metric, trControl=control, tuneLength = 10)
toc()
# -> fit.cart1: 21.75 sec elapsed
tic('fit.cart2')
set.seed(7)
fit.cart2 <- train(annual.income~.-fnlwgt, data=traindf.resampled.under, method="rpart",
                  parms = list(split = "information"), #or 'information'
                  metric=metric, trControl=control, tuneLength = 10)
toc()
# -> fit.cart2: 5.79 sec elapsed
tic('fit.cart3')
set.seed(7)
fit.cart3 <- train(annual.income~.-fnlwgt, data=traindf.resampled.both, method="rpart",
                  parms = list(split = "information"), #or 'information'
                  metric=metric, trControl=control, tuneLength = 10)
# -> fit.cart3: 12.91 sec elapsed
toc()


results.cart <- resamples(list(cart=fit.cart, cart1=fit.cart1, cart2=fit.cart2,
cart3=fit.cart3 ))
summary(results.cart)
dotplot(results.cart, metric=metric)
# -> The splitting index Information outperforms Gini
# Visualising Rpart
library(rattle)
```

```
fancyRpartPlot(fit.cart$finalModel)

# Get true test kappa score
predictions.cart <- predict(fit.cart, newdata = testdf)
predictions.cart.prob <- predict(fit.cart, newdata = testdf, type='prob')$less
predictions.cart1 <- predict(fit.cart1, newdata = testdf)
predictions.cart2 <- predict(fit.cart2, newdata = testdf)
predictions.cart3 <- predict(fit.cart3, newdata = testdf)
confusionMatrix(predictions.cart, testdf$annual.income)
confusionMatrix(predictions.cart1, testdf$annual.income)
confusionMatrix(predictions.cart2, testdf$annual.income)
confusionMatrix(predictions.cart3, testdf$annual.income)
# -> OHE = Unencoded
# -> information > Gini
# -> (Fitted on oversampled train set: fit.cart1) Kappa : 0.5436, Sensitivity : 0.7894,
Specificity : 0.8511
# -> (Fitted on undersampled train set: fit.cart2) Kappa : 0.5454, Sensitivity : 0.7856,
Specificity : 0.8614
# -> (Fitted on both over&undersampled train set: fit.cart3) Kappa : 0.5542, Sensitivity :
0.7984, Specificity : 0.8488
# -> (Fitted on original train set: fit.cart) Kappa : 0.5812, Sensitivity : 0.9388,
Specificity : 0.6018
# -> CART(0.5812) > LOGIT(0.5629) > LDA(0.5269)

# TREES: Boosting
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
#
C5.0*************************************************************************************
*************
grid.c50 <- expand.grid( .winnow = c(TRUE,FALSE), .trials=c(5,10,15,20,25), .model="tree" )

set.seed(7)
fit.C5.0 <- train(annual.income~.-fnlwgt, data=traindf,
                  method="C5.0", tuneGrid=grid.c50, metric=metric, trControl=control, verbose
= FALSE)
# -> fit.C5.0: 223.87 sec elapsed

# -> Best tuning parameters are trials = 15, model = tree and winnow = FALSE.

set.seed(7)
fit.C5.01 <- train(annual.income~.-fnlwgt, data=traindf.resampled.over,
                  method="C5.0", tuneGrid=grid.c50, metric=metric, trControl=control, verbose
= FALSE)
# -> fit.C5.01: 489.61 sec elapsed

set.seed(7)
fit.C5.02 <- train(annual.income~.-fnlwgt, data=traindf.resampled.under,
                   method="C5.0", tuneGrid=grid.c50, metric=metric, trControl=control, verbose
= FALSE)
# -> fit.C5.02: 85.45 sec elapsed

set.seed(7)
fit.C5.03 <- train(annual.income~.-fnlwgt, data=traindf.resampled.both,
                   method="C5.0", tuneGrid=grid.c50, metric=metric, trControl=control, verbose
= FALSE)
# -> fit.C5.03: 237.53 sec elapsed

results.c50 <- resamples(list(c5.0=fit.C5.0, C5.01=fit.C5.01, C5.02=fit.C5.02,
C5.03=fit.C5.03))
summary(results.c50)
dotplot(results.c50, metric=metric)

# Summary of best model
summary(fit.C5.0$finalModel)

predictions.C50 <- predict(fit.C5.0, newdata = testdf)
predictions.C50.prob <- predict(fit.C5.0, newdata = testdf, type='prob')$less
predictions.C501 <- predict(fit.C5.01, newdata = testdf)
predictions.C502 <- predict(fit.C5.02, newdata = testdf)
predictions.C503 <- predict(fit.C5.03, newdata = testdf)
confusionMatrix(predictions.C50, testdf$annual.income)
confusionMatrix(predictions.C501, testdf$annual.income)
confusionMatrix(predictions.C502, testdf$annual.income)
confusionMatrix(predictions.C503, testdf$annual.income)

# -> Unencoded > OHE
```

```
# -> (Fitted on oversampled train set: fit.C5.01) Kappa : 0.5919, Sensitivity : 0.8442,
Specificity : 0.8095
# -> (Fitted on undersampled train set: fit.C5.02) Kappa : 0.5889, Sensitivity : 0.8102,
Specificity : 0.8754
# -> (Fitted on both over&undersampled train set: fit.C5.03) Kappa : 0.5919, Sensitivity :
0.8442, Specificity : 0.8095
# -> (Fitted on original train set: fit.C5.0) Kappa : 0.6099, Sensitivity : 0.9451,
Specificity : 0.6224
# -> C5.0(0.6099) > CART(0.5812) > LOGIT(0.5629) > LDA(0.5269)


# 9.0 Compare algorithms =====================================================================
results.all <- resamples(list(logit=fit.logit5, lda=fit.lda1, cart=fit.cart, C5.0=fit.C5.0))
summary(results.all)
# Estimate Skill on Validation Dataset
# create a plot of the model evaluation results and compare the spread and the mean accuracy
of each model.
dotplot(results.all)
dotplot(results.all, metric=metric)
# -> C5.0 is the best classifier, gave the best coss-validation kappa score.
# -> In order to improve the kappa score even further, we will look into model ensembling in
the next section


# summarize Best Model
print(fit.C5.0)


# 10.0 Stacking Algorithms
================================================================================
# -> AKA Model Ensembling
# -> Combine different classifiers using model stacking
# -> In other words, combine the predictions of multiple caret models using the caretEnsemble
package.
# -> reference: https://cran.r-project.org/web/packages/caretEnsemble/vignettes/caretEnsemble-
intro.html

# Before stacking, we check correlations between predictions made by separate models,
# -> If correlation is (< 0.75), stacking is more likely to be effective.
modelCor(results.all)
# -> Since logit is very

# Specify the type of resampling, in this case, repeated 10-fold cross validation
trainControl <- trainControl(method="cv", number=10,
                             savePredictions=TRUE, classProbs=TRUE)

# Create a caretlist object (basically a list of models together)
tic('model_list_big2')
model_list_big2 <- caretList(
    annual.income~.-fnlwgt, data=traindf,
    trControl=trainControl,
    tuneList=list(
        cart<-caretModelSpec(method = 'rpart', parms = list(split = "information"),
tuneLength=10),
        lda<-caretModelSpec(method="lda", preProcess= 'BoxCox'),
        c50<-caretModelSpec(method="C5.0", tuneGrid=data.frame(.trials = 15, .model='tree',
.winnow=FALSE))
    )
)
toc()
# model_list_big2: 99.67 sec elapsed


# Visualise the results of the list of models created by the caretlist() command
results.stack <- resamples(model_list_big2)

summary(results.stack)
dotplot(results.stack)

# Let's check for correlation between the list of models (caretlist object).
# -> If the predictions for the sub-models were highly corrected (> 0:75),
#    then they would be making very similar predictions most of the time, reducing the benefit
of combining the predictions.
modelCor(results.stack)
splom(results.stack)
# -> Now create a stacked model using a simple linear model using caretStack(),
# -> caretStack allows us to use "meta-models" to ensemble collections of predictive models.
```

```
stackControl <- trainControl(method="repeatedcv", number=10, repeats=3,
                             savePredictions=TRUE, classProbs=TRUE)

set.seed(7)
fit.stack5 <- caretStack(model_list_big2, method="glm", metric=metric, trControl=stackControl)
set.seed(7)
fit.stack6 <- caretStack(model_list_big2, method="rpart", metric=metric,
trControl=stackControl)


predictions.stack.glm5 <- predict(fit.stack5, newdata = testdf)
predictions.stack.glm5.prob <- predict(fit.stack5, newdata = testdf, type='prob')
predictions.stack.rpart6 <- predict(fit.stack6, newdata = testdf)
predictions.stack.rpart6.probs <- predict(fit.stack6, newdata = testdf, type='prob')


confusionMatrix(predictions.stack.glm5, testdf$annual.income)
confusionMatrix(predictions.stack.rpart6, testdf$annual.income)
# -> Kappa : 0.6246 (model_list_big2=(cart, lda, C5.0), rpart) **WINNER!**
# -> Kappa : 0.6129 (model_list_big2=(cart, lda, C5.0), glm)


# 11.0 Visualising Results
================================================================================================
# Create a ROC plot comparing performance of all models
colors <- randomColor(count = 10, hue = c("random"), luminosity = c("dark"))
roc1 <- roc(testdf$annual.income, predictions.stack.glm5.prob, col=colors[1], percent=TRUE,
asp = NA,
            plot=TRUE, print.auc=TRUE, grid=TRUE, main="ROC comparison", print.auc.x=70,
print.auc.y=80)
roc2 <- roc(testdf$annual.income, predictions.stack.rpart6.probs, plot=TRUE, add=TRUE,
            percent=roc1$percent, col=colors[2], print.auc=TRUE, print.auc.x=70,
print.auc.y=70)
roc3 <- roc(testdf$annual.income, predictions.C50.prob, plot=TRUE, add=TRUE,
            percent=roc1$percent, col=colors[3], print.auc=TRUE, print.auc.x=70,
print.auc.y=60)
roc4 <- roc(testdf$annual.income, predictions.cart.prob, plot=TRUE, add=TRUE,
            percent=roc1$percent, col=colors[4], print.auc=TRUE, print.auc.x=70,
print.auc.y=50)
roc5 <- roc(testdf$annual.income, predictions.lda1.prob, plot=TRUE, add=TRUE,
            percent=roc1$percent, col=colors[5], print.auc=TRUE, print.auc.x=70,
print.auc.y=40)
roc6 <- roc(testdf$annual.income, predictions.logit5.prob, plot=TRUE, add=TRUE,
            percent=roc1$percent, col=colors[6], print.auc=TRUE, print.auc.x=70,
print.auc.y=30)
legend("bottomright", legend=c("stack.glm", "stack.rpart", "C5.0", "CART", "LDA", "logistic"),
col=c(colors[1:6]), lwd=2)

# 12.0 Conclusions
================================================================================================
=====
# -> STACK(0.6129) > C5.0(0.6099) > CART(0.5812) > LOGIT(0.5629) > LDA(0.5269)

# -> The kappa scores listed above are all test-kappa scores, and it was used with 100% of the
data.
# -> The dotplots are all cv-kappa scores,
#    quite frequently, some models loses out on cv-kappa scores but trumps on test-kappa
scores.

# -> I have included the runtime of the best selected models in each algorithm.

# -> Looking at the 2 performance metrics, Kappa and Sensitivity,
# -> we decided that stack.rpart (using classification trees to stack) is the best model.
# -> based on Kappa statistic, sensitivity and area under roc curve score.
```