Classifying Survey Interest from LISS data

Team 1017: Joseph Lugo (20477440), Rishi Jotsinghani (20471652), Sajeevan Uthayakumar (20475824) (all team members enrolled in STAT 441)

ABSTRACT

Our team's approach to the survey interest learning problem was to try and use multiple models which we assumed were appropriate to the problem, and choose the models which yielded the best results.

The competition provided us with 3 files, a training set, a testing set, and an additional file which contained socio-demographic info to supplement some of the data in both the training and testing set. Large amounts of time and resources were dedicated to cleaning up and preparing the data. Our goal during this phase was to perform meaningful feature engineering and feature extraction as well as prime the training and test sets so that they could be trained in R without any technical issues. Ultimately, our group tested the following models: Random Forests, MART, and Multinomial Logistic Regression. The challenges, strengths, weaknesses and results of these models will be elaborated upon in this report.

DATA CLEANUP AND PREPARATION

The data for this competition was presented in the form of three comma separated files, "avars1", "combined", and "test". The file "combined", contained information about specific surveys taken by individuals. Examples of some of the information provided in "combined" include data on the duration of the survey, the date the survey was completed, and 5 point-scale ratings on whether they thought the survey was clear, difficult or enjoyable. In addition, unique respondent IDs were also provided, hence there were instances where a respondent had completed more than one survey.

The testing data provided through the file "test" was structured in a similar way, except there were less explanatory variables to work with. The "combined" file had 15 columns, whereas the "test" file had 10 columns, this was an early indication that further down the data cleanup and preparation process our group needed to engage in feature engineering in order to make the two datasets compatible for modeling purposes.

Finally the "avars1" file contained socio-demographic information for certain respondents. By using the ID column present in all three files, we were able to merge the data from "avars1" to "combined" and "test" to further supplement the information provided in the training and testing sets. Adding the information from "avars1" also introduced many missing values in both the training and testing sets, due to the fact that some of the IDs in the "avars1" file did not exist in the training and testing sets.

The diagrams and figures below, provide a comprehensive illustration of the split of the data, missing information and class imbalances.

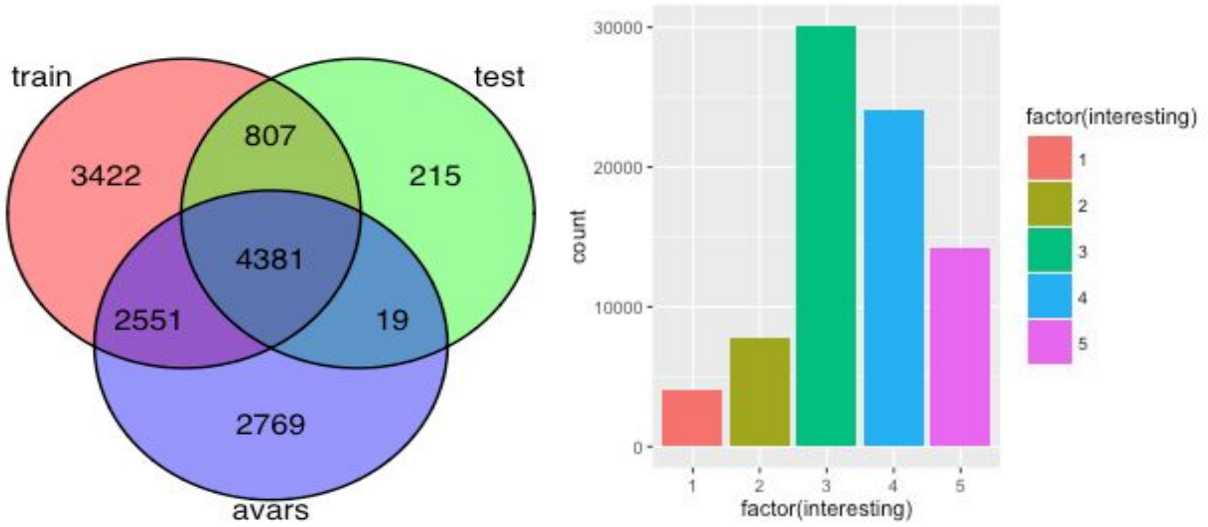

Figure 1: Left venn diagram showing overlap of IDs between data set. Right bar plot showing class imbalance

The venn diagram in *Figure 1*, displays the the overlap in IDs between the three csv files provided to us. It is clear that the majority of the IDs overlap in the three files, so this lead us to believe that the additional explanatory variables could play a key role in improving the prediction of the models. Furthermore, we noticed that there was high class imbalance in the data, namely survey interest levels of 3 and 4 heavily outnumber the rest of the interest levels. We attempted to use upsampling to balance the classes, but it proved to be ineffective.

The figure below, displays the number of NA values found in each column, after merging the “combined” file with the “avars1” file. Out of 44 variables in the training data, more than half of the variables had significantly large amounts of NA values. The problems with NA’s worsened when we tried feature engineering. The details of the new variables we derived are elaborated on later in the report, but by introducing new variables to the dataset, we also introduced more NA values.

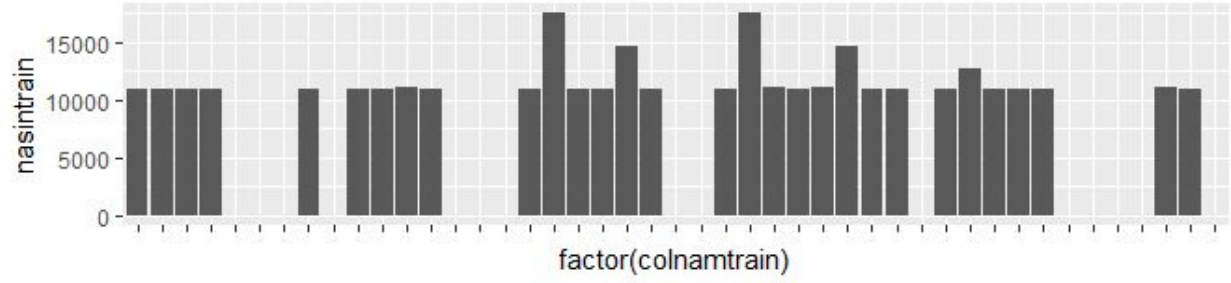

Figure 2: Bar plot showing the amount of NA values for each variable in the training data

Since the data contained many NA values that would cause problems for certain algorithms. It needed to be addressed very carefully. One option that was explored was imputation through packages in R. There are many options for filling in missing data, with many publications specializing in this very topic. The main idea behind the most simple form of imputation was to use the mean of the column to fill in the missing data for the numeric data, and use the mode for the column to fill in categorical data. Once this was done to the data, there was actually an increase in the Log Loss. This indicated that it might not be the most useful method when trying to improve predictions. This is not very surprising as it fills every missing item with the mean, so these data points would not be any different from any other of the missing data points and thus, wouldn't be providing any useful information to the models we trained.

Outliers were also problematic when trying to create predictions. When there are outliers, there is a very wide range in the data. When outliers are included in the training data, it brings with it a higher chance for misclassification. Outliers are abnormal occurrences in the data that one does not want to train on. To counteract this skewness, the log transform was applied to attempt to lessen the range and create a more compact grouping. Using a log scale is a very common practice and very effective. Boxplots were also used to identify which variables contained outliers. Once all outliers were accounted for, the numeric data was scaled using the scale function in R. This function was used to normalize the data using the normalization formula. Once the data was scaled there was a more stable distribution for all the numeric data.

With feature engineering, we wanted to create new variables that would be useful and improve our predictions. The most important set of variables to account for were the 5 missing variables in the test data (ie. interesting, enjoy, difficult, clear, thinking). In order to create an accurate prediction, these were very important to include somehow. The way we handled this was by grouping respondents by ID numbers and getting the means for each category. So every ID would have a corresponding mean for interesting, enjoy, difficult, clear, and thinking. This proved to be very effective as it greatly improved our accuracy and Log Loss scores. Other features that we crafted were things like total number of surveys that they have taken and how many surveys they have taken up to that point in time. One problem that was observed was the number of levels certain factors had. Some had too many for the algorithms to use, like startdate and enddate. To solve this problem, some variables were decomposed into multiple variables. One example of this is with the date. If the value was a character like "10/03/2010", it was decomposed into the day, month, and year to reduce the number of levels of the factor instead of completely discarding the variable. If this could not be done, the variable would have to be taken out. Principal component analysis was also attempted to reduce the dimensions but it did not decrease the Log Loss score. Even if the performance was increased, focusing on the accuracy and Log Loss was more important.

When creating models after making changes to the data, we repeatedly used all the data to train one model. Since imputation was not a reliable source of correcting data, we decided it was best to split our data into multiple sections and train based on these

sections. The 5 sections were based on the importance of variables as well as the occurrences of NAs within these columns. The training data was sectioned into the following: all observations, observations without NA values, observations with complete time data (start_hr, end_min, start_am, etc.), observations with complete demographic data (anything from the avars file), and observations without NA values and the survey mean columns (interesting_mean, enjoy_mean, etc). Note that there is overlap between these sets, as they are used to create models to predict on specific sections of the test data. Below is a pie chart showing the distribution of the test data.

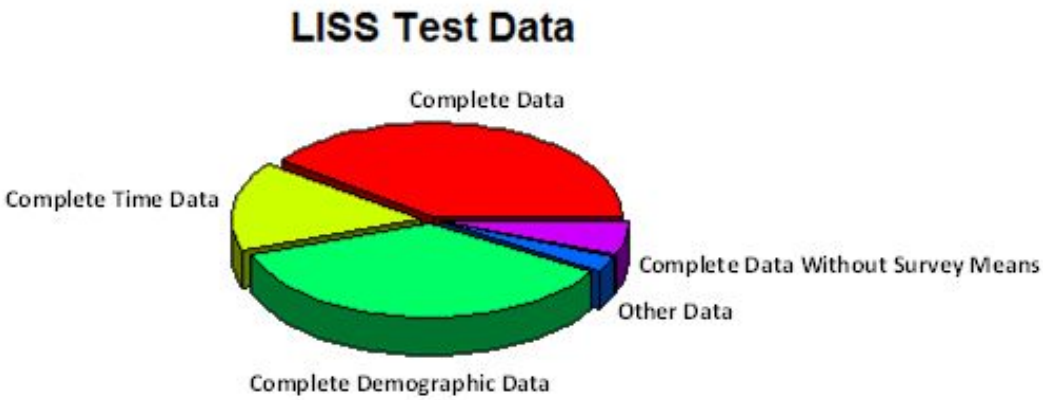

Figure 3: Venn Diagram showing breakdown of test data

MODELS AND ANALYSIS

Random Forest

The problems we first encountered when training Random Forest included that there were many observations with NA values within the training data. When training the data, the algorithm originally did not function with NA values. Another problem was that when using all the values in the data to train the model, the runtime ended up being too long to feasibly run given the resources at hand. We decided to use the caret package to create our Random Forest model, as this package integrated the randomForest package along with additional options for tuning and construction.

The main parameter to tune in a Random Forest model with the caret package is mtry, which is defined as the number of variables that are randomly selected as candidates at each split. Note that ntrees is automatically set at 500 trees and tuning ntrees would have a marginal effect on the final result once ntrees is large. When tuning mtry, our goal was to minimize Log Loss as this is the metric used to evaluate our prediction on Kaggle. When tuning randomForest to minimize Log Loss, our training function used 5-fold Cross Validation repeated 3 times, where less complex models are favoured.

With all the completed data up until this point, we were forced to tune on no more than 1000 observations, and ended up with an mtry = 68. When training the actual model that would be used to predict on our test data, we again had to limit our observations to

no more than 5000-8000 in order to avoid computation times over 30 mins. With our limited data utilization, this model was capped in terms of Log Loss. We still needed to understand how to deal with excessive runtimes and properly working around NA values. We decided to retrain our model with Parallel Processing. With Parallel Processing, we were able to diminish computation to the point to which we could use all the data and complete the model within 10 mins.

MART (aka. Gradient Boosting)

This method was appealing because of the flexibility of the algorithm. It is able to handle missing values well and although it is slower than other algorithms including Random Forests, it is quite easy to implement. It is another tree based algorithm with three main parameters. These parameters are the number of trees, the interaction depth, and the learning rate. Finding an optimal combination of these parameters contributes a good amount to improving the prediction. The number of trees will depend on what the learning rate is set to. When the learning rate is lower, the algorithm will need more trees to be able to learn properly. Interaction depth describes the number of splits for the trees and the larger the interaction depth the longer the algorithm will take.

Tuning the parameters was made possible by using the caret package in R. This package was able to tune the parameters by trying different combinations to optimize the accuracy of the model. It used repeated cross validation to find the optimal combinations of variables. Once the parameters have been tuned, training the models was quite simple. MART is able to take in all the data, and create an iterative model.

Ensemble

The ensemble approach suggests the idea of combining classifiers together to make a more accurate model. One advantage of combining a group of classifiers is that you can reduce the variance. Aggregating the models together will create a less noisy model and introduces a new set of possibilities to the classification. Another advantage is that ensembles can also handle bias in individual models. If one model was leaning more to one side, then adding another model would reduce the bias to that one side alone. Finally ensembles can aid overfitting.

The way this was implemented into our classifiers, was by using weighted averages. A function was created to optimize the weights and find which ones would minimize the Log Loss. This was done by using loops to iterate through many combinations of coefficients. Once the right weights were found, they were applied to the corresponding classifiers and this weighted average of the probabilities became the final probability matrix.

Multinomial Logistic Regression

Due to the fact that the response variable had multiple classes, we also tried to train a multinomial logistic regression model. Multinomial regression seemed suitable for this

learning problem because the predictor variables were individual specific, i.e. the model considers factors such as income, age, and various other socio-demographic information. With the help of the 'nnet' R package, we were able to train multinomial models to a certain degree of success. The model had difficulty dealing with the entire training set with all the explanatory variables. Therefore, we used some of the knowledge gained from the previous models we trained to gain insight on which explanatory variables to use. This was a good starting point, and we began to try various different combinations of explanatory variables based on intuition. Though we tried to measure the Log Loss of our model's performance using a validation set within the training data, measures of Log Loss compared to the Kaggle Log Loss scores were different. Hence we chose to evaluate our multinomial logistic model based on accuracy, for tuning purposes. Below are a few of the combinations of explanatory variables tested and their respective prediction accuracies.

interesting ~ interesting_mean + enjoy_mean + clear_mean + thinking_mean + duration + core + year : 0.5269942
interesting ~ interesting_mean + enjoy_mean + thinking_mean + core + year : .5240052
interesting ~ interesting_mean + enjoy_mean + thinking_mean + core + year + num_surveys: 0.5251261
interesting ~ interesting_mean + enjoy_mean + thinking_mean + core + year + num_surveys + duration + duration_mean : 0.5257488
interesting ~ interesting_mean + enjoy_mean + thinking_mean + core + year+ duration + duration_mean : 0.5251884
interesting ~ interesting_mean + enjoy_mean + clear_mean + thinking_mean + duration + core + year + start_am : 0.2842643
interesting ~ interesting_mean + enjoy_mean + clear_mean + thinking_mean + duration + core + year + num_surveys + past_surveys : 0.527181
interesting ~ interesting_mean + enjoy_mean + clear_mean + thinking_mean + duration + core + year + num_surveys + past_surveys + duration_mean : 0.5312909

Using the last model, we saw extremely good results with respect to the amount of time and resources spent on tuning the model, we achieved a Kaggle Log Loss of 1.16687. The performance of the multinomial logistic model can be explained by the fact that it is very apt for such modelling situations. One drawback of working with this model in R is the fact that it doesn't handle NA values properly. Hence, in the predictions submitted to Kaggle, the NA values were substituted with proportion probabilities for the respective classes. Going forward, we are looking to resolve the NA value problem and improve the accuracy and performance of this model.

CONCLUSION

Initial modelling attempts for predicting included using Naive Bayes and SVM. These methods proved to be problematic for the data that was provided. After these initial attempts, focus was shifted to MART and Random Forest. There has been a slow improvement in the score as different methods of training were introduced. Near the beginning of the project Random Forest was only assigned approximately 5000 rows as it was taking a very long time to run. As seen in *Table 1*, there have been many discoveries made in attempting to improve the Log Loss score. Once the survey means for interesting, enjoy, difficult, clear, and thinking were added according to ID, the Log Loss scores improved by 0.20676 and 0.24556 for MART and Random Forest respectively. Another idea that was attempted was dealing with the outliers using a log transformation and scaling all the numeric data.

Data	MART	Random Forest	Ensemble
Original Data	1.41983	1.45863	NA
Upsampled Data	1.42354	1.61360	NA
Adding Survey Means	1.18458	1.21307	NA
Log Transform + Scaled	1.17897	NA	1.19651
Subsets	NA	1.12606	1.11147

Table 1: Table showing results from different data inputs and algorithms

This brought marginal improvement to the MART algorithm as seen above. This is also the point when we started to explore the idea of ensemble methods. A first pass of this method used MART and Random Forest predictions without any weights. This proved to be ineffective and would be tried again at a later time. After this point we discovered how to parallelize the Random Forest computation. This is also when we decided to use the subsets of data to train and predict on these individual subsets. When we tried this approach with the Random Forest algorithm, we got a score of 1.2606, which was the most impressive result so far. Not wanting to give up on ensemble methods, we looked into using weighted averages. The function to optimize this mentioned in the data section was used to find the most optimal weights to use. This brought a score of 1.11147 which indicated that this was worth putting more work into. Moving forward, more ensemble methods will be attempted as well as trying to add the multinomial model in the list.

REFERENCES

Rosella, Laura, and Ryan Walton. "Multinomial Logistic Regression: Analysis of Multi-category Outcomes and Its Application to a Salmonella Enteritidis Investigation in Ontario." PHO Rounds: Epidemiology. 21 Feb. 2013. <https://www.publichealthontario.ca/en/LearningAndDevelopment/Events/Documents/Multinomial_logistic_regression_2013.pdf>.

Friedman, Jerome H. "Greedy Function Approximation: A Gradient Boosting Machine." (1999): n. pag. Stanford Department of Statistics. Stanford University, 19 Apr. 2001. Web. 2 Dec. 2016. <https://statweb.stanford.edu/~jhf/ftp/trebst.pdf>.

Kuhn, Max. "Building Predictive Models in R Using the Caret Package | Kuhn | Journal of Statistical Software." Journal of Statistical Software 28.5 (2008): n. pag. Building Predictive Models in R Using the Caret Package | Kuhn | Journal of Statistical Software. 22 Apr. 2008. Web. 02 Dec. 2016. <https://www.jstatsoft.org/article/view/v028i05>.

Kuhn, Max. "Parallel Processing." The Caret Package. N.p., 29 Nov. 2016. Web. 02 Dec. 2016. <https://topepo.github.io/caret/parallel-processing.html>.

Sollich, Peter, and Anders Krogh. "Learning with Ensembles: How Over-fitting Can Be Useful." Neural Information Processing Systems Conference. Neural Information Processing Systems Foundation Inc., n.d. Web. 2 Dec. 2016. <https://papers.nips.cc/paper/1044-learning-with-ensembles-how-overfitting-can-be-useful.pdf+>.