Customer relationship prediction

Tackling the KDD 2009 Cup challenge

Team Leader : Isabel Holmes

Team Members: Nabih Moukayed, Simone Zanetti & Isabel Holmes

Introduction

Predicting which customer might leave a company, or who might be ripe for a sales pitch, is a common prediction problem that a data scientist might face. The issue is that a company does not want to waste valuable time and money calling all customers in case one might decided to switch to a competitor. Neither do they want to focus sales energy on people who have no appetite for new product. Therefore it is vital that the field of potential leavers etc. be narrowed, which can be done by applying classification models on customer data.

The problem this report deals with is of exactly this nature. There were three customer actions that needed to be predicted. They were defined as:

- Appetency - a desire to buy new products or services
- Churn - a desire to leave for a competitor
- Upselling - a desire to upgrade or purchase add ons

However, there were some complications. Firstly, the data was high dimensional; it contained 50,000 observations when test and training set were combined and there were 230 variables. In addition, the outcome variables were all very imbalanced (Fig.1).

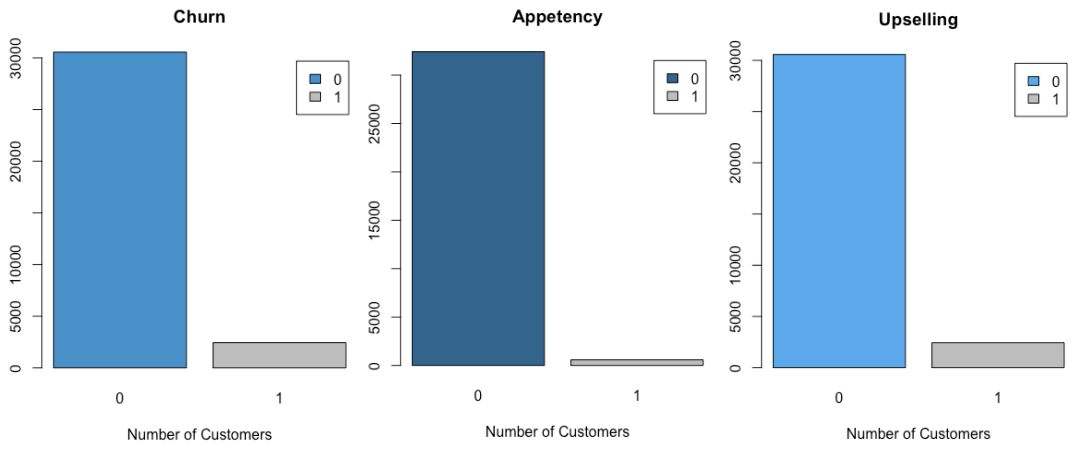


Figure 1. - Bar charts of outcome variable distribution

Finally, there was a lot of missing data. Some variables were 100% missing, and only 76 of the original 230 were below 80% missing. This made for a challenging task that required the application of advanced techniques.

Methodology

Preprocessing

We began by changing the outcome variable to 0 instead of -1 as was originally set. We did this to avoid any issues that might occur with formulas later on. We assumed that as customers with a 1 were less common, they were most likely the cases of interest i.e. the churners etc.

We then began to deal with the missing data. We started by creating a new variable that counted all NA values for each customer. We thought this might have some predictive power, for example it might indicate a person had had less contact with staff and was therefore thinking about leaving, or vice versa. We then removed any variable that was over 80% missing. We did this as there was not enough data to make a balanced variable. Now we looked to see if there was any informative missingness in the variables that were 20-80% missing. We chose this range as any new variable created by coding a 1 as a present value and 0 as NA would not be too imbalanced, as would be the case for anything more than 80% missing or vice versa. Having recoded these variables as described we performed chi-square tests to check if there was a relationship between any of these variables and our various outcomes. We could reject the null hypothesis that there was no relationship when the p value was significant. Therefore we kept only the variables for which the p values were significant and then deleted the rest. Any analysis such as this was performed on a subset of the training data so that we could keep some unseen data aside as our own test set (which I shall refer to as the validation set). Following this we had a set of columns of the variables that seemed to be significant. Rather than create a separate instance for each outcome, we noticed that they seemed to have the same variables roughly in common, so we made a sort of master file knowing using the logic that our feature selection method would remove any unnecessary variables.

At this point there was also an attempt to look for a pattern in the missingness overall in the complete raw dataset that might be helpful for feature reduction or prediction. We visualised the missingness and could indeed see a pattern; the same row seemed to be missing for several variables (see appendix 1 for missing plot). However we could not think of a way to take advantage of this pattern. This is therefore an area to perhaps revisit for future study.

Next we completed the following steps:

- Removed variables with little variance
- Removed variables that were multicollinear (over 0.6 Pearson's R) (fig.2)

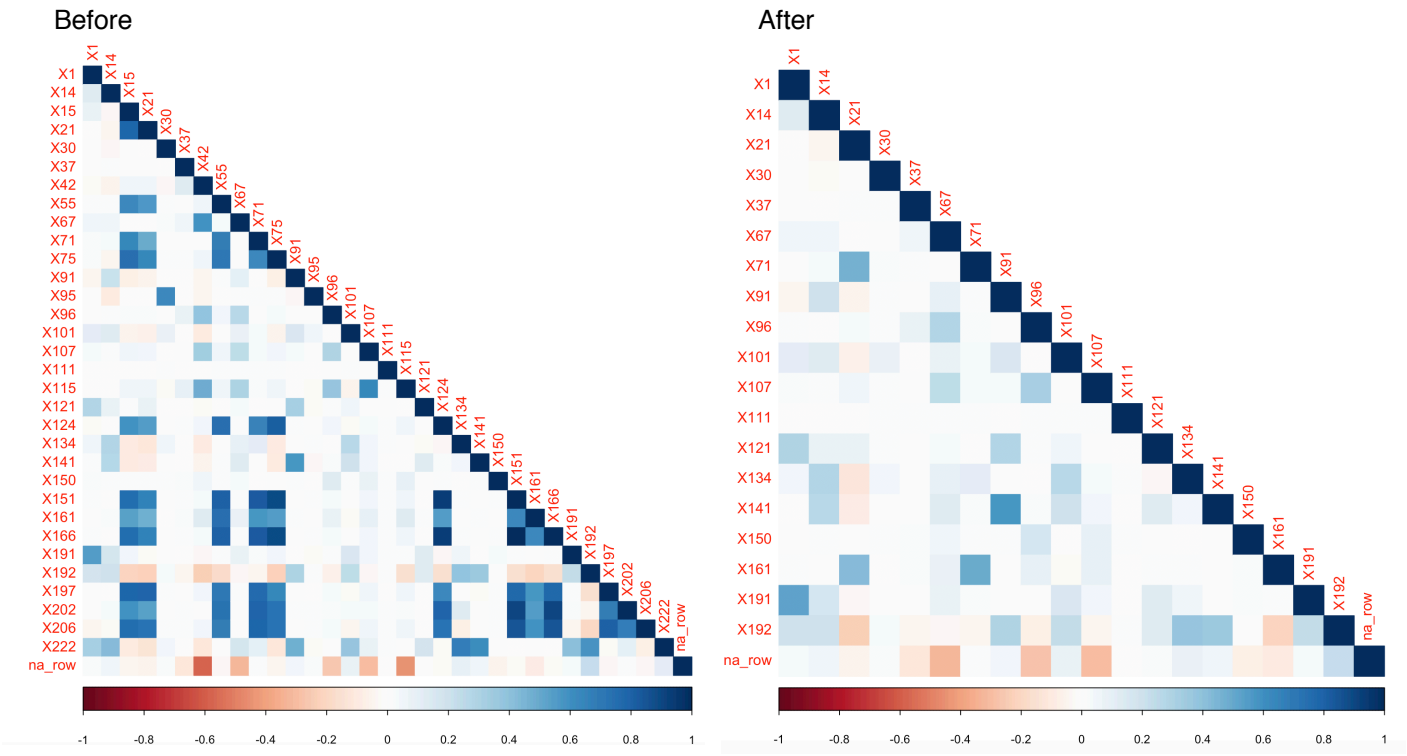


Figure 2. - Before and after corrplots. A darker colour indicates a stronger correlation.

Following this we moved on to the data imputation on our 44 remaining variable (not including informative missingness variables). We originally did this as we were using a method of imputation for numerical variables that was both computationally intensive and also prone to multicollinearity (MICE logistic and linear regression, random forests). However, we later discovered that using a simple mean impute provided better results when we built our models. This is possibly due to the nature of the final model we chose

(XGBoosted trees). In addition, we noted that the competition winners for the KDD cup in 2009 also used a mean impute (Niculescu-Mizil et al., 2009).

Moving on, we performed the following on numerical variables:

- Box-Cox transformation for skewness
- Standardisation
- Spatial sign to deal with outliers

These are all standard techniques for cleaning data and we used established packages making the work fairly straightforward.

Our next challenge was to come up with a strategy to deal with our categorical variables. Some of these had as many as 4291 levels (see fig. 3).

Categorical

	label	var_type	n	missing_n	missing_percent	levels_n	levels	levels_count	levels_percent
X29	X29	<fct>	50000	0	0.0	30	-	-	-
X90	X90	<fct>	50000	0	0.0	4291	-	-	-
X93	X93	<fct>	50000	0	0.0	4291	-	-	-
X116	X116	<fct>	50000	0	0.0	2016	-	-	-
X119	X119	<fct>	50000	0	0.0	100	-	-	-
X138	X138	<fct>	50000	0	0.0	362	-	-	-
X154	X154	<fct>	50000	0	0.0	51	-	-	-
X155	X155	<fct>	50000	0	0.0	23	-	-	-
X156	X156	<fct>	50000	0	0.0	226	-	-	-
X187	X187	<fct>	50000	0	0.0	4291	-	-	-
X189	X189	<fct>	50000	0	0.0	22	-	-	-
X190	X190	<fct>	50000	0	0.0	81	-	-	-
X204	X204	<fct>	50000	0	0.0	14	-	-	-

Figure 3. - Categorical variable information. levels_n indicates number of levels

We decided to use a method in the tidyverse package to deal with this issue. This method ranks all the levels by frequency of occurrence in the dataset, then drops all those levels below a user defined point and replaces them with a variable of the users choice. We knew we had to remove levels with fewer than 200 or so occurrences, as they created issue in the model because they were absent from folds during cross validation or from the validation set. We therefore set this as our rough cutoff and replaced anything below this frequency with a new level of ‘other’.

Feature Selection

We now performed feature selection using the relief method (Kira & Rendell, 1992). This algorithm calculates a score. This score goes down when the variable difference between instance pairs is observed in a nearest neighbour pair with the same outcome, and goes up if the opposite is true. We had to use SMOTE on our data beforehand (see below for more) to ensure results were made on balanced data. We performed this for each task outcome and selected variables with a score of over 1.65.

Addressing imbalanced data

We now used the SMOTE package to address the imbalance in our data, which would otherwise cause the models to favour the more frequently occurring and 'non-interesting' class. This package implements the SMOTE technique (Chawla etal., 2002), in which new synthetic instances of the minority class are created, in this case using k-nearest neighbour (set as 5). The package also allows the user specify how many instances of the majority class should be selected, a technique know as downsampling (when fewer instances are selected). We decided to downsample as well as creating the synthetic minority instances. This resulted in a smaller training set, but we were concerned that the alternative, higher synthetic instances of the minority, would create noise in the data. Following this we briefly reviewed the levels in the categorical variables. Using SMOTE mean that the occurrences of levels had now changed as those that were more present in the majority class had been reduced and vice-versa. We therefore corrected this issue using the same technique as before.

It should be noted that we kept aside a portion the data throughout as a validation dataset, no decisions were made using this data and we did not perform SMOTE on it. We did this so that we could have an accurate reflection of how are model might perform on real-world unbalanced data.

Models

At this point the tasks were split into three; churn, appetency and upselling. Trapezoidal AUC was used for model selection, using the following formula:

$$
\begin{aligned}
TrapezoidAUC &= Area\ Triangle + Area\ Trapezoid \\
&= \frac{(1 - Spec) \times Sens}{2} + \frac{(Sens + 1) \times Spec}{2} \\
&= \frac{Sens - SensSpec}{2} + \frac{SensSpec + Spec}{2} \\
&= \frac{Sens + Spec}{2}
\end{aligned}
$$

Churn

The first model we tried was an elastic net logistic regression. This is a technique that uses both L1 penalty of Lasso regression (allows coefficients to become 0) and the L2 penalty of Ridge regression (minimises coefficients). Together these allow the coefficients to be reduced meaning they are useful for datasets with many features. This method was selected as dummification of our categorical variables (each level becoming a new variable containing 1 or 0), with some having over 20 levels, resulted in many features. However, this method was not particularly effective so we abandoned it in favour of models that do not require dummification and were more powerful.
Our next attempt was random forest. Here decision trees are created on bagged datasets, with only a certain percentage of predictors (chosen at random) used in each tree. We used k-fold cross validation with 5 folds to find the best value for the number of variables to be chosen at each split. Again, results were disappointing. To assess these initial results we looked at the kappa of the validation dataset as well as sensitivity and specificity.

Finally we decided to try a boosted model, XGBoost. This method is a form of gradient boosting in which trees attempt to predict the error of the previous learner. We again used k-fold cross validation with 5 folds to select the best parameters, tuning our model. We were using the caret implementation of XGBoost, and this allows the user to instruct the model to generate 50 random options for each parameter. We also added a weight scaling argument in an attempt to further mitigate class imbalance. Finally we found a custom metric function online. This function allowed the model to prioritise the minimisation of the distance between specificity and sensitivity when training. Since our data is imbalanced we thought this was a better approach than training to maximise accuracy. Indeed, we ran the model three times; once training to maximise kappa, once to maximise accuracy and

the third to minimise distance. The final model had the best results on the validation dataset using area under curve (AUC) (see table 1 for parameters).

Final parameters for Churn task	
Number of rounds for boosting	733
Maximum depth	9
Eta (learning rate)	0.0639
Gamma	3.18
Ratio of columns selected each round	0.484
Minimum weight for continued splitting	1
A subsample of training instances (prevents overfitting)	0.5268921

This led to a an AUC on the training data of 82% (figure 4 for ROC curve) and 59% on the validation (figure 5 for ROC curve). Trapezoid AUC was 55% on the validation data.

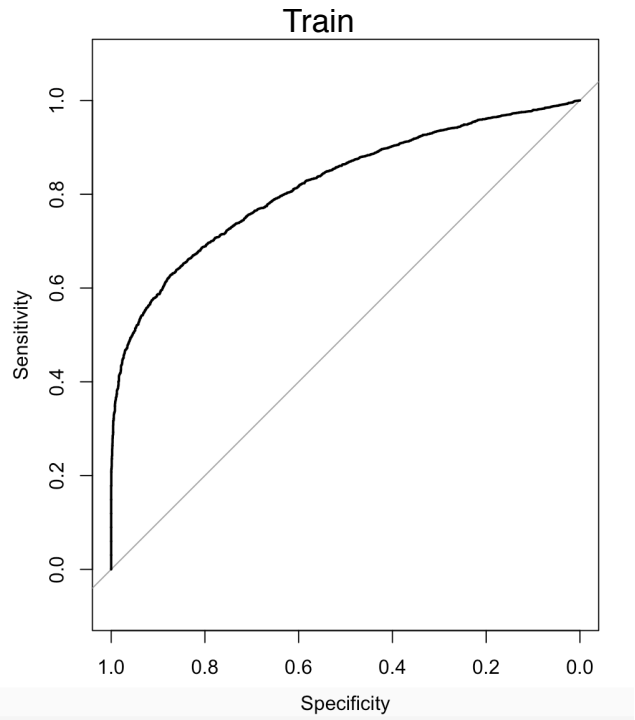


Fig. 4 - Churn ROC curve on training data

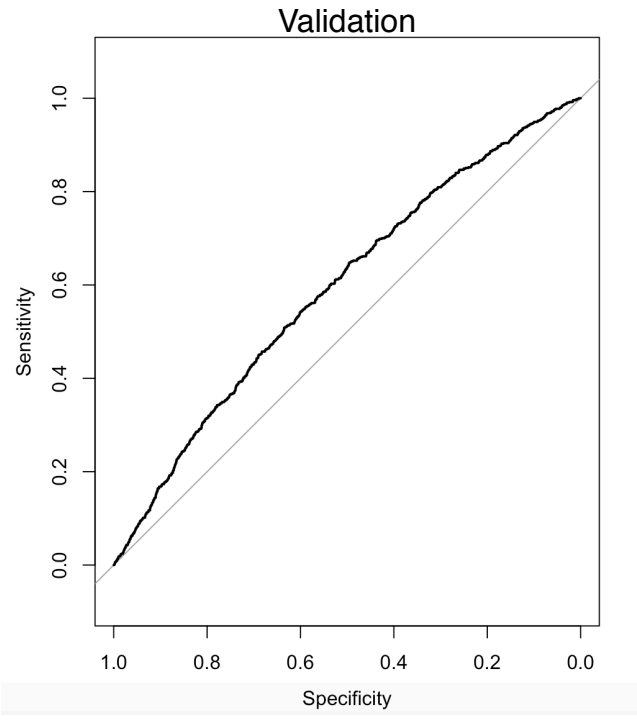


Fig. 4 - Churn ROC curve on validation data

Upselling

For the upselling task we proceeded in much the same way with similar results, i.e. elastic net and random forest were not as successful as the XGBoost model. Once again we used the custom metric to minimise distance and this produced the best results (see table 2 for parameters after tuning).

Final parameters for Upselling task	
Number of rounds for boosting	105
Maximum depth	9
Eta (learning rate)	0.446
Gamma	1.58
Ratio of columns selected each round	0.541
Minimum weight for continued splitting	0
A subsample of training instances (prevents overfitting)	0.82

The resulting AUC was 85% on the train set and 67% on the validation set. Trapezoidal AUC was 60% on the validation set.

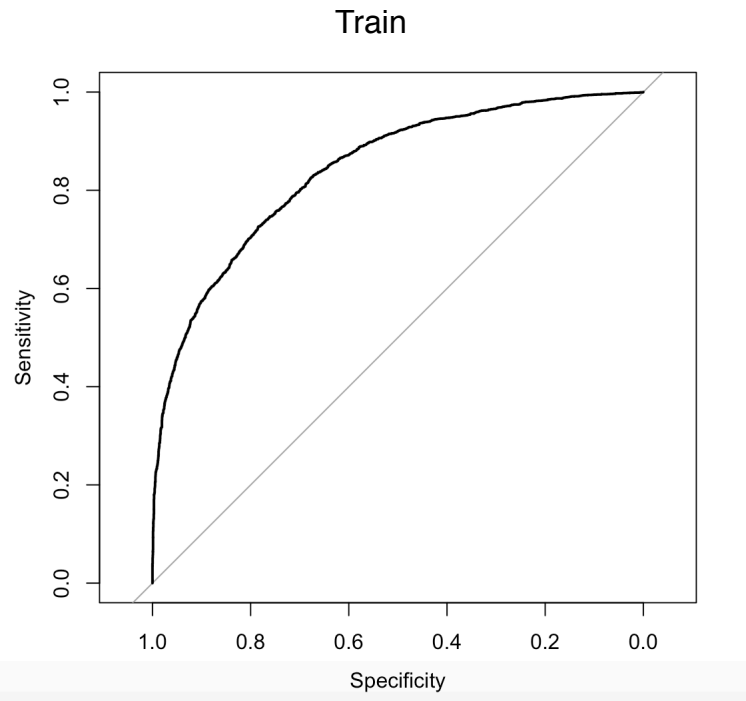


Fig. 6 - Upselling ROC curve on training data

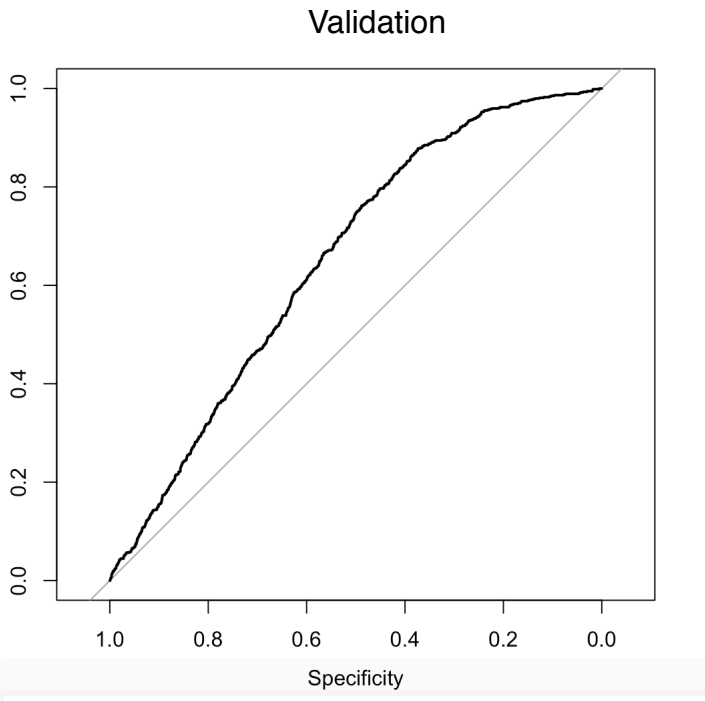


Fig. 7 - Upselling ROC curve on validation data

We also experimented with using all predictors in the model, not just those chosen by the relief method. However, as shown in figure 8, this resulted in a poor model.

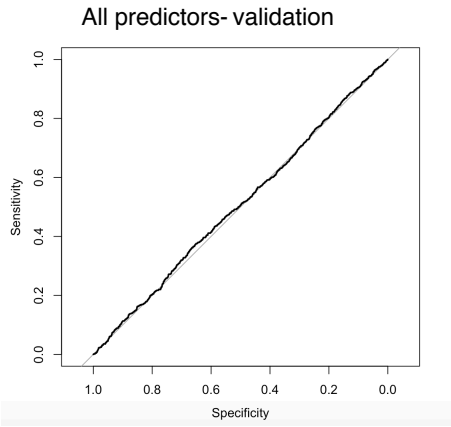


Fig. 8 - Upselling ROC curve on validation data all predictors

Appetency

Again, the same methods were applied for appetency, with the addition of a logistic regression. We also experimented with post-processing here on the regression models, which improved specificity at the cost of AUC. As the goal of the task was to maximise AUC, we abandoned this approach. The final model that gave the best trapezoidal AUC was the XGBoost model, AUC 54% (see table for parameters).

Final parameters for Upselling task	
Number of rounds for boosting	776
Maximum depth	7
Eta (learning rate)	0.2560025
Gamma	1.537886
Ratio of columns selected each round	0.6191189
Minimum weight for continued splitting	1
A subsample of training instances (prevents overfitting)	0.5819389

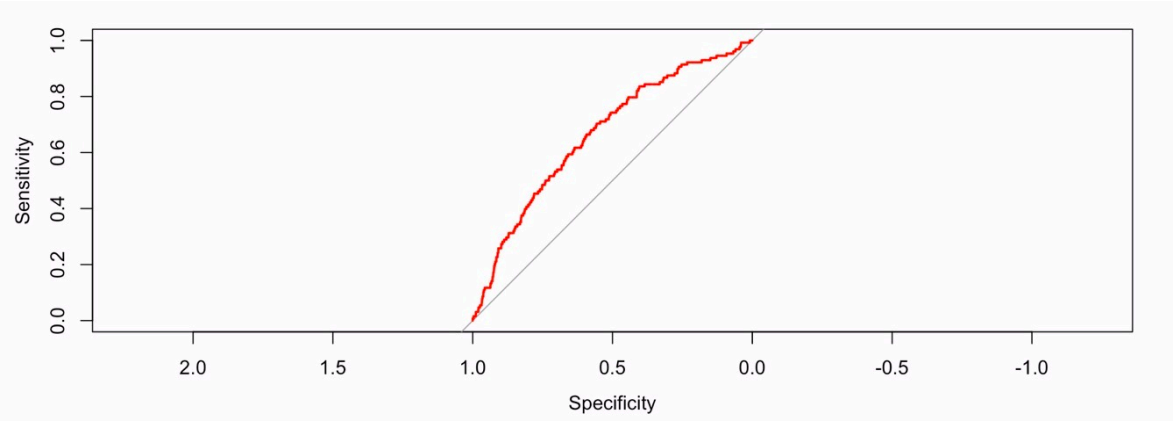


Fig. 9 - Appetency ROC curve on validation data

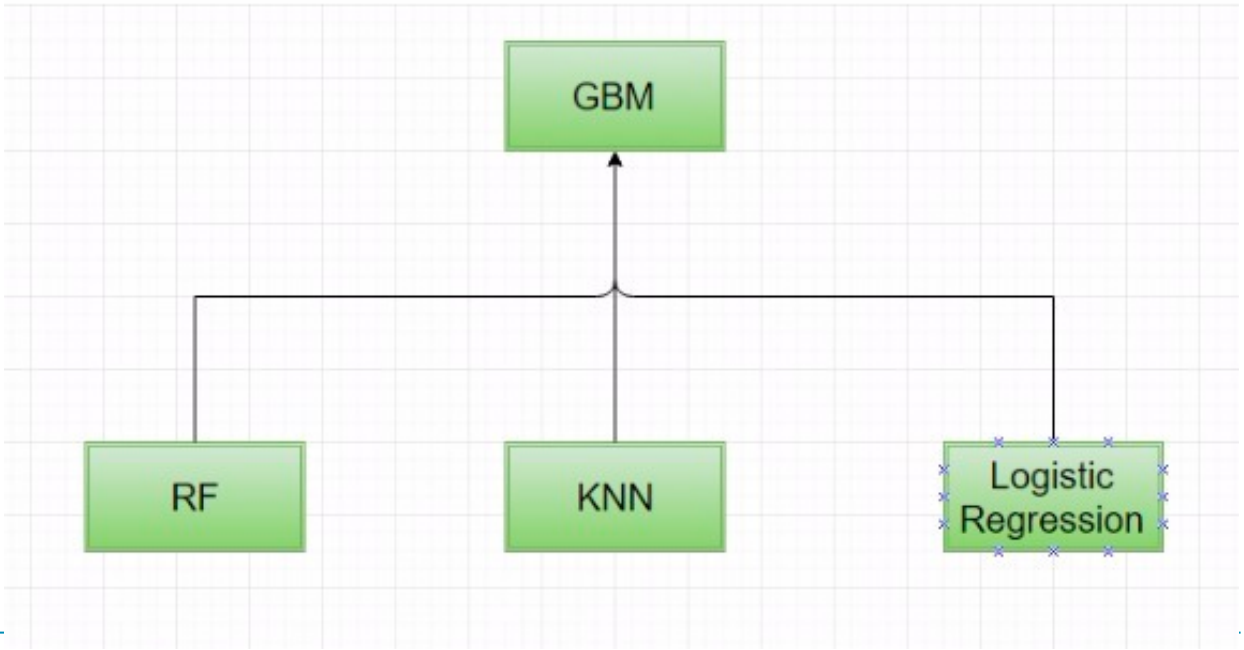
Conclusion

In conclusion, we found that the XGBoost model was consistently the better performer when it came to maximising trapezoidal AUC. However, validation results were far from ideal. There are several ways we could improve this moving forward.

Firstly, we could experiment with different methods of addressing class imbalance. We could create a bigger training set by increasing the minority synthetic samples and decreasing the amount by which we downsampled. We could also have used a more simple approach, just downsampling or upsampling without SMOTE. These approaches could result in noise data however.

Next we could try Adaboost, which is a boosting method that adjusts weights based on accuracy of the learner (Freund & Schapire,1996), meaning the algorithm focuses on targeting misclassified observations. We did not try this as we simply ran out of time. The same was the case for models such as KNN and support vector machines. An attempt to implement the latter resulted in R crashing, even on the cluster. We did also briefly try implementing a neural network (see appendix 2 for example code), but were beset with errors and decided to focus on the boosted model that seemed to be producing the best results.

In addition, we could also create an ensemble model that uses multiple algorithms as shown in the image below.



Graphic taken from here: https://www.analyticsvidhya.com/blog/2017/02/introduction-to-ensembling-along-with-implementation-in-r/

Here a random forest model, inn approach and a logistic regression all feed into a final gradient boosted model before producing a final prediction. Again, lack of time meant this was not possible.

On the appetency task, we noticed that our sampling approach meant that the validation set was substantially bigger than the training set. We therefore adjusted the split here (it was 75% train 25% val before SMOTE). This increased validation trapezoidal AUC in the final model due to more data available for training (4% boost). This could have been pushed further and rolled out across other models as an easy boost to AUC. Indeed, we feel that a lack of training data in part due to the sampling issue might have been one of our main issues.

We also noticed that our relief method was unstable in that it produced different results each time. It's possible that increasing the number of iterations might solve this issue. Further experimentation is required. Finally, we could have experimented further with post-processing. The elastic net which used post processing was the best performer on the appetency task, and it's possible this would have improved all models. Once more a lack of time prevented this.

All in all, this task has been an excellent learning experience for us and we have seen first hand just how challenging and complex dealing with messy data can be. It is shame that we could not work to further improve our AUC, but we have many ideas for the future and the experimentation which ultimately ate into our time has provided invaluable experience.

NB: *Code for certain models may be found in different files due to author differences. For example, the code for elastic net regression is only present in the appetency code file despite being applied elsewhere. A further improvement on this task would have been better workflow and code organisation, as we were all new to group work like this.*

References

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, *16*, 321–357. https://doi.org/10.1613/jair.953

Freund, Y., & Schapire, R. E. (1996, July). Experiments with a new boosting algorithm. In *icml* (Vol. 96, pp. 148-156).

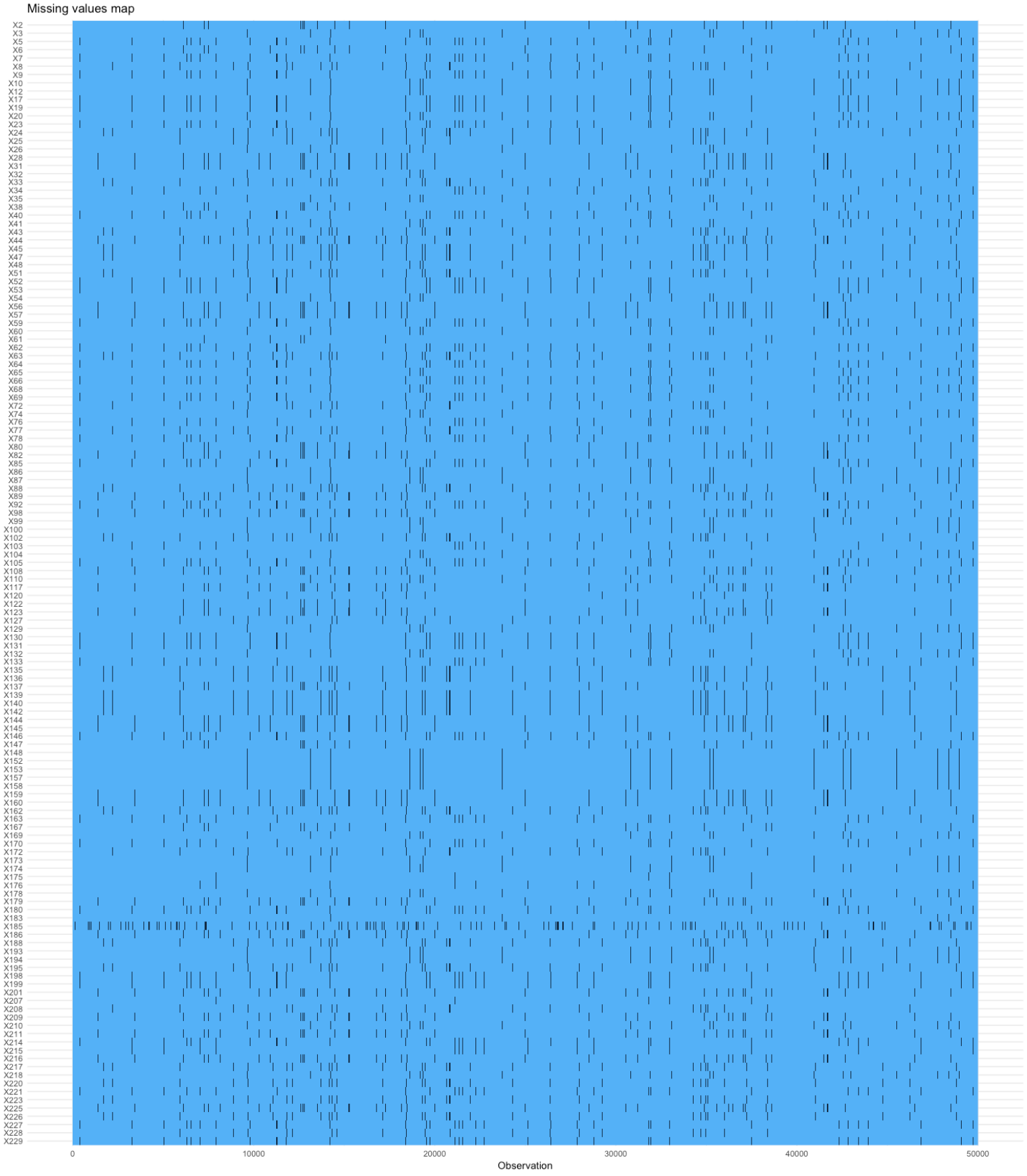
Kaushik, S. (2017). How to build Ensemble Models in machine learning? (with code in R). Retrieved April 1, 2019, from https://www.analyticsvidhya.com/blog/2017/02/introduction-to-ensembling-along-with-implementation-in-r/

Kira, K., & Rendell, L. A. (1992, July). The feature selection problem: Traditional methods and a new algorithm. In *Aaai*(Vol. 2, pp. 129-134).

Niculescu-Mizil, A., Perlich, C., Swirszcz, G., Sindhwani, V., Liu, Y., Melville, P., … Singh, M. (2009). Winning the KDD cup orange challenge with ensemble selection. *The 2009 Knowledge Discovery in Data Competition (KDD Cup 2009) Challenges in Machine Learning, Volume 3*, *7*, 21. Retrieved from http://proceedings.mlr.press/v7/niculescu09/niculescu09.pdf

Appendix

Appendix 1 - Missing plot of variables over 80% missing



Appendix 2 - Neural network code example

```
library(caret)
library("doParallel")
library("RANN")

#setwd('/Users/isabelholmes')

setwd('/home/iholm001/r_codes')

data <- read.csv("churn_train.csv", header = TRUE, na='?')

data$churn <- ifelse(data$churn == 1, 'yes',
                     ifelse(data$churn == 0, 'no', NA))

data$churn <- as.factor(data$churn)

#Start parallel process with half the number of cores
cl <- makeCluster(detectCores())
registerDoParallel(cl)

fiveStats <- function(...) c(twoClassSummary(...),
                         defaultSummary(...))

## For 10 Cross validtions
ctrl <- trainControl(method = "cv",
                     number = 10,
                     classProbs = TRUE,
                     summaryFunction = fiveStats,
                     verboseIter = TRUE,
                     allowParallel = TRUE)

# define the grid for tuning, (this is just an example, real grids contains more values)
grid = expand.grid(size = c(3,6,9,12,14,16),
                   decay=c(0.01,.1,.5,.2,.001))

nnetFit <-  caret::train (x=data[,-22],
                          y=data[,22],
                          method = "nnet",
                          trControl = ctrl,
                          tuneGrid = grid,
                          metric='Kappa')

ff_glimpse(data)


nnetFit

write.table(nnetFit, 'nn_reg')
```

Appendix 3 -Elastic Net Appetency coefficients

(Intercept)	-1.11605446
(Intercept)	.
X112_RAYp	-0.01790752
X112_ZI9m	0.48992610
X112_6fzt	0.59847224
X149_UYBR	0.85709715
X168_Mtgm	0.69757370
X189_Other	.
X189_sYC_	.
X189_hAFG	0.09852854
X189_wMei	-0.11761962
X189_haYg	.
X189_zm5i	.
X189_y6dw	0.62797921
X204_me75fM6ugJ	-0.38578703
X204_7M47J5GA0pTYIFxg5uy	.
X204_Kxdu	.
X204_DHn_WUyBhW_whjA88g9bvA64_	-0.06133036
X22_1	0.42241779
X182_0	-0.27556607
X1	-0.38470849
X67	0.95690247
X91	.
X96	0.25492052
X101	1.32080837
X192	0.62342079
na_row	.
X70	.