MACHINE LEARNING COURSEWORK 2

Team Taka – Chris Journeay (Team Lead) - Fabio Caputo – Imanol Belausteguigoitia

Abstract

Prediction of Churn, Appetency and Upselling from Orange Customer Data

Contents

Table of Tables

Table of Figures

Introduction

The challenge for this coursework was to build models to predict three potential customer outcomes from a sample set of data from the CRM of a large mobile telecom provider. The three customer outcomes that we need to predict are:

1. Churn – This is the outcome where a customer ceases their relationship with the provider (i.e.: the customer does not renew their contract and switches to another provider)
2. Appetency – This is the outcome where a customer shows a propensity to buy a product or service
3. Upselling – This the is outcome where a customer purchases additional services over and above their current spending (i.e.: newer package, more minutes, more data, etc.)

These outcomes are binary in nature – either a customer churns or does not churn. Our work is fundamentally a challenge to build models for classification of customers based on the available dataset.

Dataset Overview

The data, provided by French mobile phone provider Orange, consists of 3 files.

1. train_x.csv – A csv file with all of training observations. There are 33,001 observations with 230 variables.
2. train_y.csv – A csv file with the targets for the training data. There are 33,001 observations with 3 variables for the three outcomes churn, appetency and upselling. Positive outcomes are represented by the number '1' whereas negative outcomes are represented by '-1.'
3. test_x.csv – A csv file with the test observations. It has the same 230 variables as the train_x set but with 16,999 observations. This makes it just over half the size of the training data set.

Measurement of success

The results will be evaluated with the so-called Area Under Curve (AUC). It corresponds to the area under the curve obtained by plotting sensitivity against specificity by varying a threshold on the prediction values to determine the classification result.

The AUC is calculated using the trapezoid method. In the case when binary scores are supplied for the classification instead of discriminant values, the curve is given by {(0,1), (tn/(tn+fp), tp/(tp+fn)), (1,0)}. This is the formula that we use for our AUC calculations.

The performances are evaluated according to the arithmetic mean of the AUC for the three outcomes (churn, appetency. and upselling).

Data Exploration

Obfuscation

The data, as expected, has been completely anonymised to protect customer identities, but it has also been completely obfuscated. There are no recognisable titles in the column, the content of the categorical variables is unintelligible in any language and the numerical values also appear to have been transformed to make their original values inaccessible.

The implications of this obfuscation are that we cannot bring any business intelligence to our analysis and we are completely beholden to the analysis that we can bring to bear on the data to derive any meaning.

Dimensions and Data Variance

The data set for training is quite sizable, not so much in observations but in predictors. The challenge of building a model around 230 predictors is not insignificant. Moreover, there is a fair amount of variance in the factor data. Factor columns show a range of factor levels that range between unitary levels and unique.

It is clear we need to reduce the overall dimensionality of our data set to get a solid prediction and this will shape our approach to the factor data.

Class Imbalance

The target data is very heavily class imbalanced. An analysis of the observations in the train_Y values shows the positive rate for each outcome to be very small.

Outcome	Observations	Positive Outcomes	Positive Outcome %
Churn	33,001	2440	7.39%
Appetency	33,001	584	1.77%
Upselling	33,001	2431	7.36%

This has implications for our approach to training the data and ensuring the we train to detect the smaller class.

Missing Values

An analysis of the columns shows that there are columns with significant amounts of missing data. There are 18 columns are completely blank whereas others are so low on content that they might as well be empty; only 76 columns have completion rates above 10%.

Conclusions from the data exploration

Out initial challenge with this data set it so address he challenge of dimensionality. This will be the focus of the pre-processing stage. Subsequent step will look to account for the class imbalance.

Data Pre-processing

The focus of the pre-processing stage is to reduce the dimensionality of the data. We will focus on reducing the number of predictors, imputing the missing data and processing the data within the remaining predictors to reduce variance.

Note on data pre-processing

From this point on assume that we are applying the same pre-processing steps for each outcome and essentially delivering multiple training sets. This makes sense as there is no guarantee that there is a single set of transformations that will work for all outcomes, so our approach will be to handle each outcome as separate challenge, with three challenges addressed in a single document.

It can also be assumed that at each stage of the pre-processing, the associated test data is manipulated in an identical manner (i.e.: same columns are removed) as in their specific training set. Overall this produces a set of unique training and test set pairs.

Eliminate columns using a missing data threshold

We analysed the predictors to look at the number of missing values in each column, expressed as a percentage of observations in each column. Sorting the data from most complete to least complete we can see the following percentages.

```
> print(sorted_by_non_empty$x) #prints out the percentages
  [1] 100.0000000 100.0000000 100.0000000 100.0000000 100.0000000 100.0000000 100.0000000
  [8] 100.0000000 100.0000000 100.0000000 100.0000000 100.0000000 100.0000000 100.0000000
 [15] 100.0000000 100.0000000 100.0000000 100.0000000 100.0000000 100.0000000  99.9939396
 [22]  99.7030393  99.7030393  99.7030393  99.2272961  98.6121633  98.6121633  96.0789067
 [29]  90.0184843  90.0184843  90.0184843  90.0184843  90.0184843  90.0184843  90.0184843
 [36]  90.0184843  90.0184843  90.0184843  90.0184843  90.0184843  90.0184843  90.0184843
 [43]  90.0184843  90.0184843  90.0184843  90.0184843  90.0184843  90.0184843  90.0154541
 [50]  89.6003151  89.6003151  88.9700312  88.9700312  88.9700312  88.9700312  88.9700312
 [57]  88.9700312  88.9427593  88.9427593  88.9427593  88.9427593  88.9427593  88.9427593
 [64]  85.5610436  85.5610436  85.5610436  72.2917487  55.2771128  55.2771128  49.0863913
 [71]  49.0863913  47.5834066  43.0380898  42.0805430  25.5204388  25.5174086   7.4937123
 [78]   3.0817248   3.0817248   3.0817248   3.0817248   3.0817248   3.0817248   3.0817248
 [85]   3.0817248   3.0817248   3.0817248   3.0817248   3.0817248   3.0817248   3.0817248
 [92]   3.0817248   3.0817248   3.0817248   3.0817248   3.0817248   3.0817248   3.0817248
 [99]   2.9938487   2.9938487   2.9938487   2.9938487   2.9938487   2.9938487   2.9938487
[106]   2.9938487   2.9938487   2.9938487   2.9938487   2.9938487   2.9938487   2.9938487
[113]   2.9938487   2.9938487   2.9938487   2.9938487   2.9938487   2.9938487   2.9938487
[120]   2.9938487   2.5484076   2.5484076   2.5484076   2.5484076   2.5484076   2.5484076
[127]   2.5484076   2.5484076   2.5484076   2.5484076   2.5484076   2.5484076   2.5484076
[134]   2.5484076   2.5484076   2.5484076   2.5484076   2.5484076   2.5484076   2.5484076
[141]   2.5484076   2.5484076   2.5484076   2.5484076   2.5484076   2.5484076   2.5484076
[148]   2.2847792   2.2847792   2.2847792   2.2847792   2.2847792   2.1181176   2.1181176
[155]   2.1181176   2.1181176   2.1181176   1.7120693   1.7120693   1.7120693   1.6878276
[162]   1.6878276   1.6878276   1.6666162   1.6666162   1.6666162   1.6666162   1.6666162
[169]   1.3575346   1.3575346   1.3575346   1.3575346   1.3575346   1.3575346   1.3575346
[176]   1.3575346   1.3575346   1.3575346   1.3575346   1.3575346   1.3575346   1.3575346
[183]   1.3575346   1.3575346   1.3575346   1.3575346   1.3575346   1.3575346   1.3575346
[190]   1.3575346   1.3575346   1.3575346   1.3575346   1.3575346   1.3575346   1.3423836
[197]   1.3423836   1.3423836   1.3423836   1.3423836   1.3211721   1.3211721   1.3211721
[204]   1.0848156   1.0848156   1.0848156   0.8575498   0.7060392   0.6908882   0.4908942
[211]   0.3545347   0.3545347   0.0000000   0.0000000   0.0000000   0.0000000   0.0000000
[218]   0.0000000   0.0000000   0.0000000   0.0000000   0.0000000   0.0000000   0.0000000
[225]   0.0000000   0.0000000   0.0000000   0.0000000   0.0000000   0.0000000
```

Figure 1 - Training data by completion percentage of observations

Only 76 columns have a completion percentage above 10% and only 69 columns are more than 50% complete.

We set the threshold to 70% as our completion rate and drop any columns that fall below that cut of point. The following columns are retained.

```
> print(columns_to_keep)
 [1]  11  14  29  42  73  90  93 111 112 116 119 128 154 155 168 187 190 204 212 224 165 125
[23] 156 196 138 118 149 143   1  15  30  71  75  91  94  95 113 124 134 141 151 171 191 202
[45] 206 219 222 230 101  58  79  55  67 161 166 189 192  37  81  96 107 115 150  21 121 197
[67]  84
```

Figure 2 - Columns to retain after applying a missing data threshold of 70%

This leaves us with 67 columns of data to take to the next stage.

Centre and scale of numerical values

We used the caret package preprocess functions to address the challenge of huge variance in the numerical data, we centred and scaled the data. This facilitated the following check of near zero variance.

Near-zero variance check

We ran an analysis of the 67 columns with centred and scaled numeric data using the caret package to check for near-zero variance in the observations. We then adopted the recommendations from the package and eliminated another 10 columns leaving us with a data set of 57 predictors.

Impute missing numerical data tackle skewness.

We used the mlr package to impute the missing numbers for every numerical column. This was quite successful and eliminated any remaining NA values in the training set. Despite the centring and scaling before this step, there remain some outliers in the data set that need to be addressed at a future point. This will be done with a spatial sign transformation just before training the model.

We then used the caret package to apply a Yeo-Johnson transformation to the numerical data to address any skewness to the data sets - - there were some variables with skewness > 1.5 and kurtosis > 9.

Impute the missing factor data

At this point we took two approaches to the missing factor data.

Approach 1: Replace NA with “None”

For this approach we assumed that there was meaning in the NA values of the factor data and converted any NA values to a “None” level. This could give meaning to “blanks” in the training set that could enrich any models

Approach 2: Replace NA with Mode Imputation

We assumed that there was no meaning in the NA values of the factor data and imputed the NA values with the mode for the column

This gave us two distinct training sets to take the next stage of pre-processing.

Addressing the variance of the factor data

There was significant variance that remained in the factor data. Several data columns had more than 100 levels with several containing over 2000 different factor levels. We needed to reduce the variance, so we looked to bin the levels in each factor predictor. Our challenge was that the data had been obfuscated, there was no heuristic approach we could take to bin the values. To address this, we took 2 different approaches based on the response rate in each level.

Approach 1: We binned the data using positive response rate quartiles

The idea behind this approach was to reduce the number of factor levels for each column down to no more than 5 and increase the signal for any levels that could provide a positive response.

For each column we calculated the positive response rate for each factor level. The response rate is simply a calculation to look at what percentage of positive responses that a level contributes to the total.

This gave us a distribution of positive response rates. We performed a Box-Cox transformation on that distribution to even out any skewness. We then broke that distribution into quartiles and assigned a value to each level based on that quartile.

The factor level was then replaced with its corresponding quartile value. Any zero response levels received their own value (‘5’).

This drastically reduced the dimensionality in the factor columns and provided a measure of the positive signal strength for each factor level.

Approach 2: We binned the data using k-means clustering of the response rate

This approach is similar to the one above but we used k-means clustering to determine the number of bins for each column.

We use the positive and negative response rates for each factor level as inputs and then ran multiple k-means scenarios. We used the elbow method to determine the optimum number of neighbours using the within groups sum of squares metric. An example of the graphs we used are below

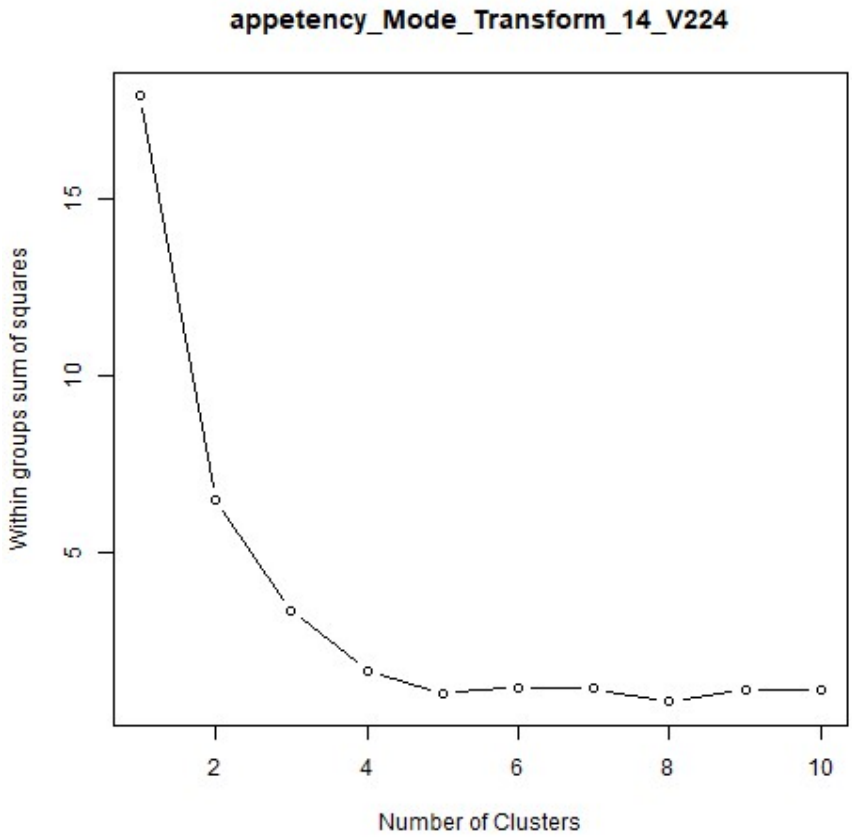

Figure 3 - Sample elbow method evaluation of k-means for appetency outcomes

K-means clustering was run again using the optimum k values. Each factor level was replaced with its associated cluster.

Both approaches were executed on both imputed factor training sets from above. This gave us a total of 4 training data sets and their matching test sets.

Do keep in mind that this is per outcome – so this gives us 12 training data sets overall.

Using Relief to remove additional columns

Even with the above efforts to reduce dimensionality, our training sets still have 57 predictors. We ran the U the Relief package in R on the four data sets to remove additional columns by computing the relief statistical test. During this process, each target variable has been shuffled several times to break any link between the predictors and the label. Relief statistics are calculated for every permutation of each predictor before generating a distribution of those values.

The next step is a hypothesis test to estimate whether the a given predictor yields a relief statistic significantly different than one with no link with the response. For any values over a certain value, we reject the null hypothesis of the two samples coming from the same underlying population. We chose this relatively large alpha value (5%) in order to retain as much information as possible. Given the class imbalance for all outcomes, this seemed to be a sensible decision.

This effectively doubled our training and test sets again.

We were able to reduce the predictors the following for each training set

Training Set	Churn	Appetency	Upselling
NA Replaced Factors binned by quartile	41	45	37
NA Replaced Factors binned by k-means	38	45	45
Mode Imputed Factors binned by quartile	47	45	42
Mode Imputed Factors binned by k-means	40	35	41

Table 1 - Training Set Predictors after Relief

This was helpful in reducing predictors however we retained the pre-Relief as well as the post-Relief training data sets for our models.

Using PCA to reduce the data set dimensionality

Principal Component Analysis is a commonly used data reduction technique. This method seeks to find linear combinations of the predictors, known as principal components (PCs), which capture the most possible variance. It creates components that are uncorrelated. Because PCA seeks linear combinations of predictors that maximise variability, it will naturally first be drawn to summarising predictors that have more variation. This means that the PC weights will be larger for the higher variability predictors on the first few components.

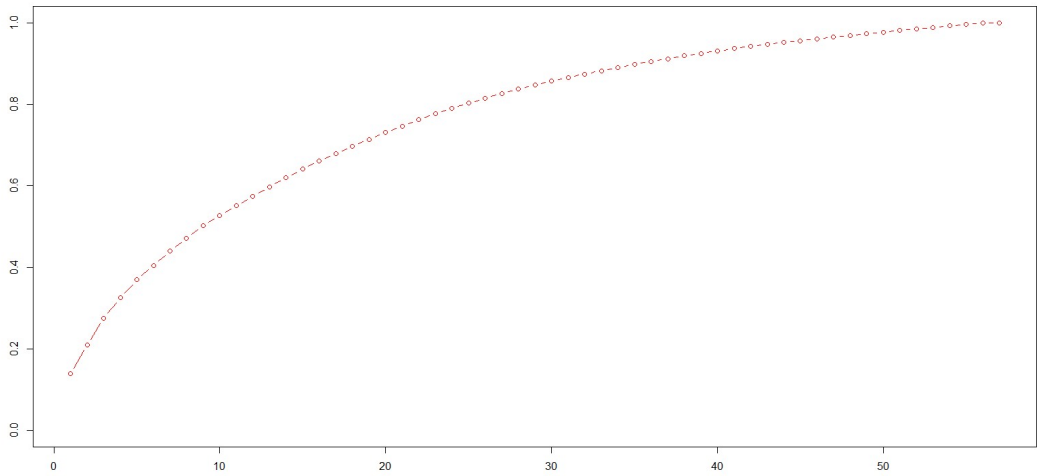

Figure 4 - The cumulative % variance explained for a Churn training set

Principal component analysis has been experimented for the prediction of each of the three response variables

Model Training

Models for training

The pre-processing of the data gave us 8 different models for training for each outcome, Churn, Appetency and Upselling. Each training set also had its own associated testing set to use after model selection.

Training Set	Factor Impute	Factor Bin Method	Reduced using Relief
Set 1	NA -> None	K-Means	No
Set 2	NA -> None	Response Quartile	No
Set 3	Mode	K-Means	No
Set 4	Mode	Response Quartile	No
Set 5	NA -> None	K-Means	Yes
Set 6	NA -> None	Response Quartile	Yes
Set 7	Mode	K-Means	Yes
Set 8	Mode	Response Quartile	Yes

This provided a rich set of options to feed into the models so we could best tune our performance.

Training Approach

Training was done using the caret package which enabled us to leverage the multiple training methods and tuning grid options.

Training methodology

To address outliers in the data, a spatial sign transformation was applied prior to modelling.

To address the severe class imbalance, data sampling for training was done using SMOTE.

All training runs for models were trained to maximise the ROC to determine the best model.

All data was trained using 3 sets of Repeated Cross Validation for every option in the training grids.

Data was also optionally subjected to Principle Component analysis before training.

We selected the best Kappa value to determine the best model from the training grids. The Sensitivity and Specificity associated with that Kappa value were used to estimate the Area Under Curve.

Area Under Curve was estimated as follows. In the case when binary scores are supplied for the classification instead of discriminant values, the curve is given by {(0,1), (tn/(tn+fp), tp/(tp+fn)), (1,0)}.

Training Models

Training was run using the following models

- Bagged Trees
- Random Forest
- XGB
- Gradient Boosting Machines
- Support Vector Machines

Bagged Trees

An ensemble of decision trees models is used to predict the class of a new sample. Each model has equal weight in the ensemble, and each can be thought of as casting a vote for the class it thinks the new sample belongs to. The total number of votes within each class are then divided by the total number of models in the ensemble to produce a predicted probability vector for the sample. The new sample is then classified into the group that had the most votes, and therefore the highest probability.

Random Forest

As with bagging, each tree in the forest casts a vote for the classification of a new sample, and the proportion of votes in each class across the ensemble is the predicted probability vector. However, the most significance difference lies in the idea behind randomly sampling predictors during training is to de-correlate the trees in the forest.

The main tuning parameter is the number of randomly selected predictors to choose from each split, denoted as mtry.

Boosting

Boosting generates a sequence of weak classifiers, where at each iteration the algorithm finds the best classifier based on the current sample weights. In the standard version of the algorithm, samples that are incorrectly classified in the kth iteration receive more weight in the (k+1) iteration, while samples that are correctly classified receive less weight in the subsequent iteration. This means that samples that are difficult to classify receive increasingly larger weights until the algorithm identifies a model that correctly classify these samples. Therefore, each iteration of the algorithm is required to learn a different aspect of the data, focusing on regions that contain difficult-to-classify samples. The overall sequence of weighted classifiers is then combined into an ensemble and has a strong potential to classify better than any of the individual classifiers. Boosting can be applied to any classification technique, but classification trees are a popular method for boosting since these can be made into weak learners by restricting the tree depth to create trees with few splits. Since classification trees are a low bias/high variance technique, the ensemble of trees helps to drive down variance, producing a result that has low bias and low variance.

There are several versions of the boosting algorithm, including extreme gradient boosting using ensembles of trees.

Support Vector Machines

Support Vector Machine is an extension of the soft margin classifier that results from implicitly enlarging the feature space in a specific way, using kernels - i.e. functions used to quantify the similarity of two observations. The scope is to accommodate a non-linear boundary between the classes. A popular choice to enlarge the feature space is the radial kernel, alongside with polynomial kernels. SVMs are one of the most effective and flexible machine learning tools available.

Training Outcomes for Churn

Using the Best Kappa calculated AUC Trapezoid method the top 5 performing models for Churn are as follows:

Training Set	Model	AUC Trapezoid
Training Set 4	xgb trees model	0.928540528
Training Set 4	xgb trees model with PCs (85% VAR) as features	0.935632969
Training Set 4	xgb trees model with additional PCs (85% VAR) as features	0.99254326
Training Set 4	GBM	0.917190346
Training Set 4	GBM model with PCs (85% VAR) as features	0.930880009

Table 2 - Best performing training models for Churn

Training Commentary

+10 models have been built to predict the Churn target variable.

Reducing the dimensionality of the data set based on the outcome of the Relief test has proven to deteriorate the model performance. On the contrary, using principal components has been the key factor to improve the results on cross validation. In particular, the first 30 components, explaining 85% of the variance in the data, have been selected.

All best outcomes have been obtained using the Training Set 4 and employing tree based algorithms and PCA. In particular, oblique trees models achieved the best results.

While CART and C4.5 classification trees are the most widely used, there has been extensive research in this area and many other proposals for tree-based models. Other types of splits can be employed. In particular, Breiman et al. (1984) introduced the idea of splitting on a linear combination of the predictors. These oblique trees may be beneficial when the classes are linearly separable, which traditional splits have a difficult time approximating. Menze et al. (2011) also discusses tree ensemble models with oblique trees.

This approach has proven particularly beneficial to predict churn when combined with boosting, random forest and bagged trees - the average area under the curve for such models is ~ 98%. However, the increased dimensionality of the model, result of the addition of the PCs, could be more prone to overfitting. This can be experimented when comparing the predictions generated against the true labels of the test set.

Support Vector Machines performed overall worse than the aforementioned models.

Selected Model for Churn

The training set selected for Churn was Training Set 4

- NA Values imputed with mode
- Factors binned using quartiles of the positive response rate
- No reduction using Relief

The model selected for Churn was Extreme Gradient Boosting with the addition of principal components with the following tuning parameters.

- max_depth =7
- nrounds = 100

- eta = 0.5
- colsample_bytree =0.6
- min_child_weight = 1.527525
- 30 principal components (explaining 85% of the variance in the data)

Training Outcomes for Appetency

Using the Best Kappa calculated AUC Trapezoid method the top 5 performing models for Appetency are as follows:

Training Set	Model	AUC Trapezoid
Training Set 1	Support Vector Machines	0.9744103
Training Set 2	Bagged Trees with CART	0.9693434
Training Set 3	Support Vector Machines	0.9657741
Training Set 4	Bagged Trees with CART	0.9656101
Training Set 4	Random Forest	0.9654416

Table 3 - Best performing models for Appetency

Training Commentary

The Appetency Outcome provided a real challenge given the small positive outcomes. The training sets reduced using Relief performed less well on most instances – but not all. No training set emerged as a "best choice" in this instance suggesting that all pre-processing options removed useful information at some point. Generally, adding Principle Components reduced performance in all models. In summary, less processed data performed better with reduction by Relief performing worse overall.

Support Vector Machines and Bagged Trees models performed best overall for this outcome providing 4 out of the top 5 performing models. Bagged trees were selected as the chosen model as it was the best performing with the most complete test set.

Selected Model for Appetency

The training set selected for Appetency prediction was Training Set 2:

- NA Values transformed into None
- Factors binned using quartiles of the positive response rate
- No reduction using Relief

The model selected for Appetency was Bagged Trees with the following tuning parameters

- ntrees = 50

Training Outcomes for Upselling

Using the Best Kappa calculated AUC Trapezoid method the top 5 performing models for Appetency are as follows:

Training Set	Model	AUC Trapezoid
Training Set 4	Random Forest	0.9475356
Training Set 4	Random Forest	0.9471240
Training Set 4	GBM	0.9395923
Training Set 4	GBM	0.9367030
Training Set 4	Bagged Trees	0.9313487

Table 4 - Best performing models for Upselling

Training Commentary

For each of these algorithms several models were deployed. In this sense, special attention was placed in the performance of Kappa and Sensitivity and Specificity over other metrics. The process was focused on changing the tuning grip parameters to get to the most successful models. The class imbalance was fixed with SMOTE technique, which is exactly the same used for each model run for upselling.

Random Forest gave the best results of the techniques used, followed by Gradient Boosting Machines and Bagged Trees.

Selected Model for Upselling

The training set selected for Appetency prediction was Training Set 4:

- NA Values imputed with mode
- Factors binned using quartiles of the positive response rate
- No reduction using Relief

The model selected for Upselling was Random Forest with the following tuning parameters

- mtry = 8