MATH289C: Final Exam

Mitesh Gadgil (A53095373), Saurabh Kulkarni (A53099844)

UCSD, March 2016

Contents

1 Introduction

1.1 Problem Statement

We are provided with an anonymized dataset containing both categorical and numeric variables available when the claims were received by BNP Paribas Cardif. All character/string type variables are categorical. There are no ordinal variables. The "target" column in the train set is the variable to predict. It is equal to 1 for claims suitable for an accelerated approval and 0 otherwise.The given data comprises of two types of datasets:

- Training Data
- Testing Data

The train data consists of 114393 observations and 132 variables. A target column is also given mentioning whether the claim is suitable for an accelerated approval or not. We have to use these 114000 data points to train our classifier so as to classify the data points i.e. claims given in the testing data set appropriately. We will perform some preliminary data analysis on the given training data to get some information out of it and look at possible features for the classifier.
The test set consists of 114393 observations and 132 variables.

1.2 Challenges

This problem is a big data classification problem and the challenges of this problem are as follows:

- The data contains 131 features. Some features are dominant during classification while some are not. We are not given any additional information so as to which features are crucial.
- Since it is very difficult to work with all 133 features, we need to use some technique to reduce the number of features to n. This reduces the complexity of the problem by reducing the 133- dimension space to n-dimensional space. Lasso/ Group Lass can help in doing this.
- Some features have a lot of missing values. We need to device a technique to either impute these missing values or to reject them altogether.
- About 21 of the given variables are categorical variables. That means they take some random 'character' value *eg.: A,B,C,D... or eg:* U_1, U_2, U_3*... etc.* Essentially we have no clue as to what these value mean or how important they are for training the classifier.

1.3 Evaluation Metric

The trained classifier will be evaluated using the logloss function defined as:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i\log(p_i) + (1-y_i)\log(1-p_i)\right)$$

where N is the number of observations, *log* is the natural logarithm, y_i is the binary target, and p_i is the predicted probability that y_i equals 1. Log Loss heavily penalises classifiers that are confident about an incorrect classification. For example, if for an observation, the classifier assigns a very small probability to the correct class then the corresponding contribution to the Log Loss will be very large. Naturally this is going to have a significant impact on the overall Log Loss for the classifier. The bottom line is that it's better to be somewhat wrong than emphatically wrong. There are at least two approaches to dealing with poor classifications:

- Examine the problematic observations relative to the full data set. Are they simply outliers? In this case, remove them from the data and re-train the classifier.
- Consider smoothing the predicted probabilities using, for example, Laplace Smoothing. This will result in a less "certain" classifier and might improve the overall Log Loss.

2 Preliminary Data Analysis

We will perform some basis data processing on the data to observe patterns in numerical data, to tackle the missing values and the various categorical variables.

2.1 Dataset Description

We first see that the dimension of the datasets were as follows

- Training Data: [114321, 133]
- Testing Data: [114393, 132]

The train data consists of 114393 observations and 133 variables (1 for ID, 1 for target and 131 variables). The test set consists of 114393 observations and 132 variables (1 for ID and 131 variables).
The first variable is called 'ID' and is present in both the train and test set. The second variable in the train set is called 'target'. This is the variable that needs to be predicted in the test set and is therefore not present in the test set. The other variables are named V1 to V131 and are available in both the train and test set. These variables can be categorical or quantitive. Some categorical variables have many levels (for example more than 18211 levels). There are no duplicate observations in both the train and test set.

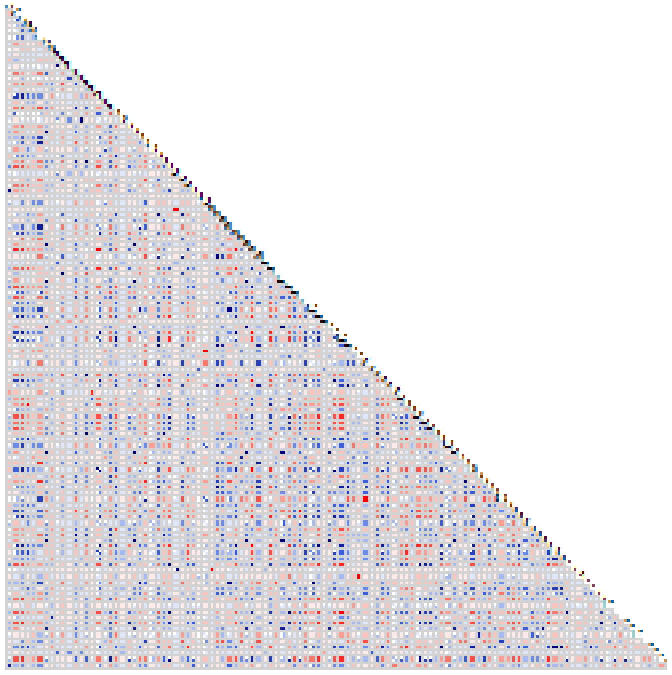
The variable target in the training dataset can take the values 0 = not suitable for accelerated approval,1 = suitable for accelerated approval. In the test set you need to predict the probability that the claim is suitable for accelerated approval.
Of the 114321 claims, **87021 are suitable for accelarated approval**. Thus if I have no further information, I would say the the **prior probability for an accelarated approval is 0.7611987**. Lets try that as a benchmark. This is also the Maximum Likelihood Probability that the target class of a datapoint = 1 (assuming sample observations are IID).

2.2 Numerical Variables

Most continous variables have a minimum value of zero, except for the variables v14 (-0.00001), v20 (1.52), v21 (0.1062), v25 (0.04), v46 (0.07), v54 (0.01), v63 (0.05), v65 (0.66), v68 (1.50), v70 (0.43), v87 (0.87), v89 (0.02), v123 (0.02),

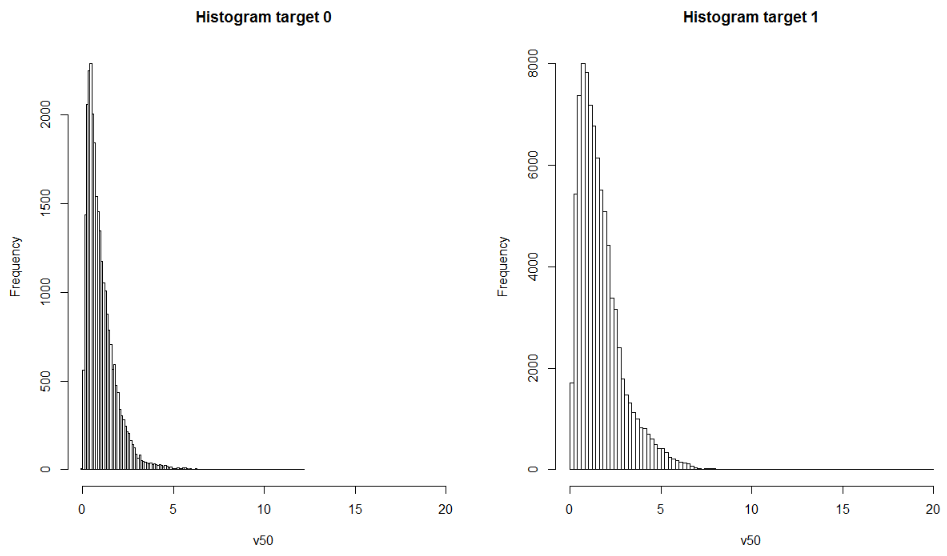
For most variables the maximum value is 20.00, except for the following variables: v10 (18.534), v12 (18.711), v21 (19.2961), v28 (19.85), v32 (17.56), v38 (12.00), v39 (19.92), v44 (19.83), v61 (18.85), v62 (7.00), v70 (19.82), v72 (12.00), v77 (15.97), v86 (17.56), v87 (19.84), v90 (6.31), v92 (8.92), v94 (19.02), v95 (9.07), v98 (19.06), v103 (18.78), v120 (10.39), v123 (19.69), v126 (15.63) and v129 (11.00). We observe that the values are never higher than 20.00.

We can find how correlated each variable is using *corrgram* plot. This gives us a visual representation of the correlation of each variable with each other. In other words, it gives us the lower half of of the correlation matrix.

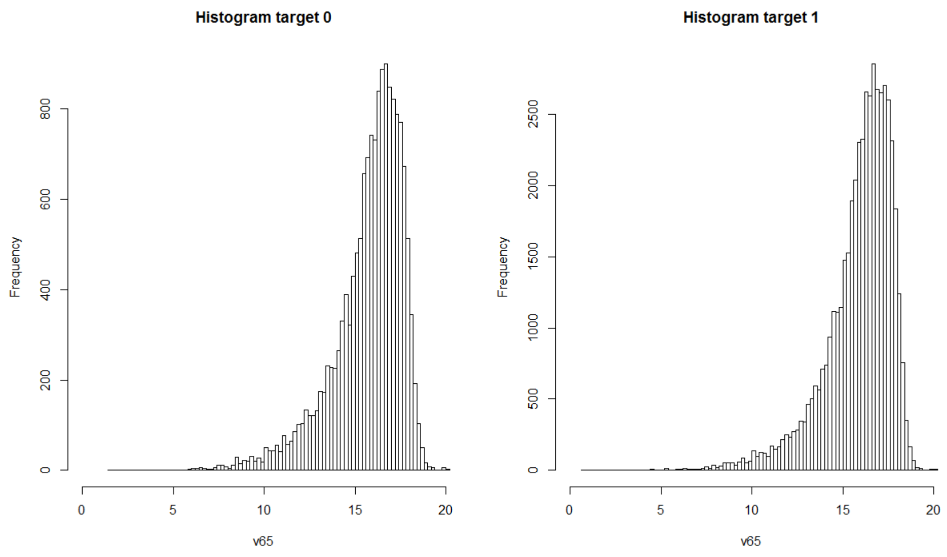


As shown in the figure above, some variables are highly correlated. Can this be used to reduce the number of variables? Positive correlation is shown in blue and negative in red. Higher the correlation magnitude, darker the hue. This is a very important analysis as some of the classification algorithms we have used like like logistic regression assumes uncorrelated variables we must remove the highly correlated ones. We went ahead and calculated the values of maximum correlation of each variable with the other. This was done to identify which variables are highly correlated to remove them from further analysis. However, it seems that there are no duplicate variables i.e. no two variables convey the same information, hence we can't drop variables easily without further analysis.

We now observe the distribution of the each feature for different values of the target. Our main aim is to observe the distribution of each feature for different values of target. If the feature is a dominant variable during classification then we would observe two distinct distribution. If the feature is a weak feature then we would observe similar distributions.
Note: Since the number of datapoints with target 1 is almost 3.18 times the number of datapoints with target=0. Hence the histograms of target 1 will be scaled by approximately 3.18 times the ones generated with target 0.



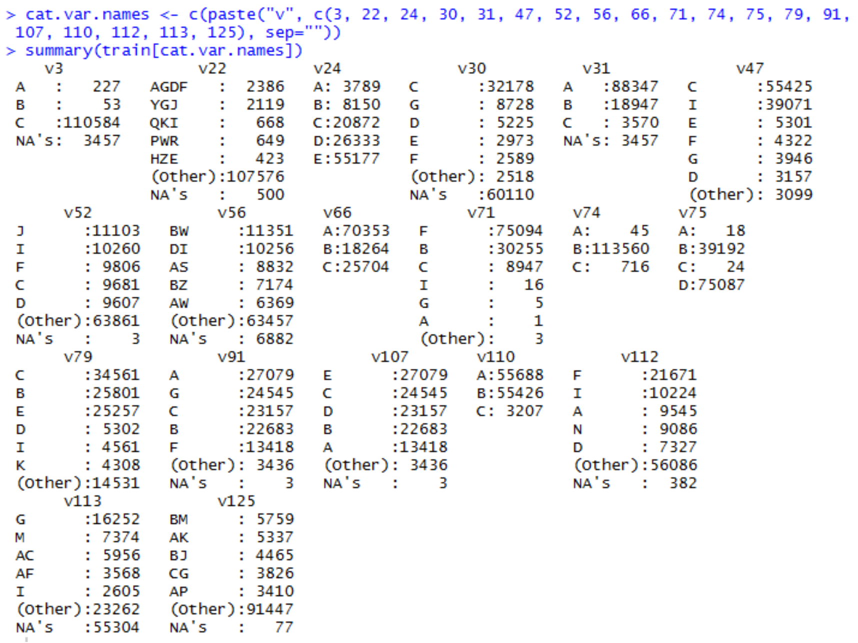
In the above figure we clearly observe that: For different values of target, the feature v50 has a different distribution. Hence we can state that the data from *target = 0* and *target = 1* come from different populations. Hence the variable **v50** clearly plays an important role in classification.



Here in the above distribution we observe that the two distributions are almost same. Hence we conclude that this variable is not a good feature for classification purposes.

2.3 Categorical Variables

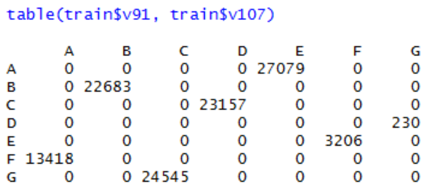
We now observe the trends the categorical variables follow. We first summarize the list of categorical variables and the possible values they take. The following snapshot of the summary of categorical variables is as follows:



```
> cat.var.names <- c(paste("v", c(3, 22, 24, 30, 31, 47, 52, 56, 66, 71, 74, 75, 79, 91,
 107, 110, 112, 113, 125), sep=""))
> summary(train[cat.var.names])
     v3                v22           v24               v30              v31               v47       
 A   :   227   AGDF     :  2386   A: 3789   C       :32178   A   :88347   C       :55425  
 B   :    53   YGJ      :  2119   B: 8150   G       : 8728   B   :18947   I       :39071  
 C   :110584   QKI      :   668   C:20872   D       : 5225   C   : 3570   E       : 5301  
 NA's:  3457   PWR      :   649   D:26333   E       : 2973   NA's: 3457   F       : 4322  
               HZE      :   423   E:55177   F       : 2589                G       : 3946  
               (Other):107576               (Other): 2518                D       : 3157  
               NA's     :   500             NA's    :60110                (Other): 3099  
      v52             v56          v66             v71           v74          v75       
 J       :11103   BW      :11351   A:70353   F       :75094   A:    45   A:   18  
 I       :10260   DI      :10256   B:18264   B       :30255   B:113560   B:39192  
 F       : 9806   AS      : 8832   C:25704   C       : 8947   C:   716   C:   24  
 C       : 9681   BZ      : 7174             I       :   16              D:75087  
 D       : 9607   AW      : 6369             G       :    5                       
 (Other):63861   (Other):63457              A       :    1                       
 NA's    :    3   NA's    : 6882             (Other):    3                       
      v79             v91                v107          v110            v112      
 C       :34561   A       :27079   E       :27079   A:55688   F       :21671  
 B       :25801   G       :24545   C       :24545   B:55426   I       :10224  
 E       :25257   C       :23157   D       :23157   C: 3207   A       : 9545  
 D       : 5302   B       :22683   B       :22683             N       : 9086  
 I       : 4561   F       :13418   A       :13418             D       : 7327  
 K       : 4308   (Other): 3436   (Other): 3436             (Other):56086  
 (Other):14531   NA's    :    3   NA's    :    3             NA's    :  382  
      v113            v125      
 G       :16252   BM      : 5759  
 M       : 7374   AK      : 5337  
 AC      : 5956   BJ      : 4465  
 AF      : 3568   CG      : 3826  
 I       : 2605   AP      : 3410  
 (Other):23262   (Other):91447  
 NA's    :55304   NA's    :   77  
```

In the above figure, we see what are the different values taken by each categorical variable as well as the corresponding count in the training dataset.

One obvious question we intend to ask is are any of the two variables correlated. That is if there is a clear relationship between any of the two variables we can eliminate them easily. To observe this we plot the correspondence matrix of the categorical variables. Note that variable **v91** and **v107** have the same values but different labels.



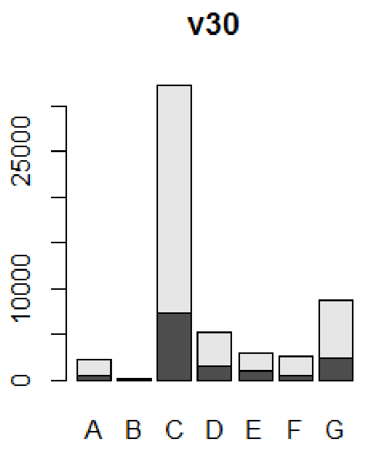
```
table(train$v91, train$v107)

        A     B     C     D     E     F     G
  A     0     0     0     0 27079     0     0
  B     0 22683     0     0     0     0     0
  C     0     0     0 23157     0     0     0
  D     0     0     0     0     0     0   230
  E     0     0     0     0     0  3206     0
  F 13418     0     0     0     0     0     0
  G     0     0 24545     0     0     0     0
```

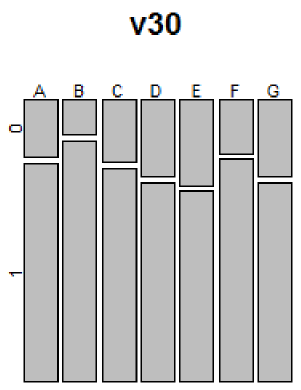
In the above table we can clearly see that there is a one-one mapping between **v91** and **v107** as shown in the following table:

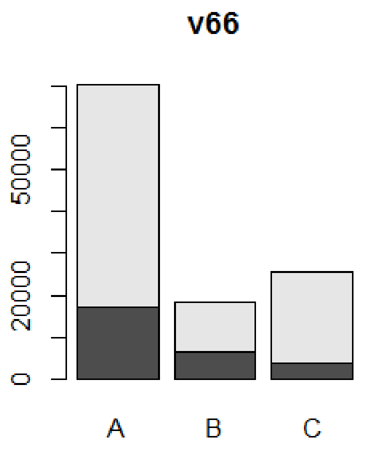
A of v107	F of v91
B of v107	B of v91
C of v107	G of v91
D of v107	C of v91
E of v107	A of v91
F of v107	E of v91
G of v107	D of v91

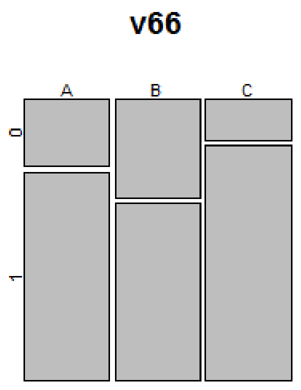
Hence we conclude that these two variables must be the same and we can eliminate one to avoid redundancy.

Along with this we also plot, we also plotted the bar plots of all the categorical variables. Two sample barplots have been shown below.







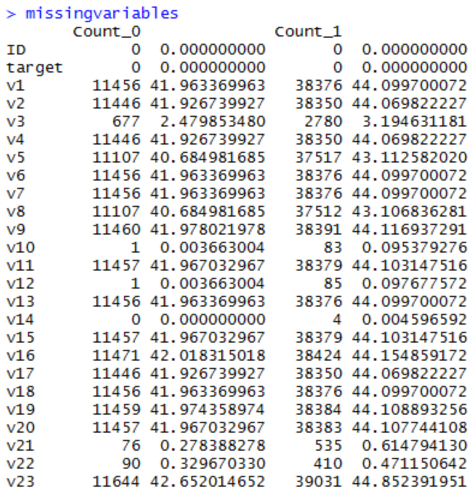


By observing all barplots we conclude that: Variables v3, v30, v31, v47, v66, v71, v74, v75 look the most promising as predictors.

2.4 Missing Values

The summary data shows that the categorical variables have no missing values, but they have blanks. Replace those blanks with NA to indicate missing values. Of the total points, **17756 observations have no missing values** The first two variables "ID" and "target" have no missing observations. There are a few other variables (v24, v38, v47, v62, v66, v71, v72, v74, v75, v79, v110, v129) without missing observations. For most variables the proportion of missing variables is 43%. There is one variable (v3) with more than 50% missing and 17 variables with a proportion of missingsness less than 10%. We also observe that for all the variables with missings, the percentage missings is lower in the target = 0 group compared to the target = 1 group. This information can be used to predict the probability that target = 1 in the test set. Only the continuous variables v38, v62, v72 and v129 have no missing values.

The screenshot below indicates the number of missing values in each variable along with corresponding percentage of missing variables, for each value of target. (This is part of the table)



```
> missingvariables
       Count_0                Count_1
ID           0  0.000000000       0  0.000000000
target       0  0.000000000       0  0.000000000
v1       11456 41.963369963   38376 44.099700072
v2       11446 41.926739927   38350 44.069822227
v3         677  2.479853480    2780  3.194631181
v4       11446 41.926739927   38350 44.069822227
v5       11107 40.684981685   37517 43.112582020
v6       11456 41.963369963   38376 44.099700072
v7       11456 41.963369963   38376 44.099700072
v8       11107 40.684981685   37512 43.106836281
v9       11460 41.978021978   38391 44.116937291
v10          1  0.003663004      83  0.095379276
v11      11457 41.967032967   38379 44.103147516
v12          1  0.003663004      85  0.097677572
v13      11456 41.963369963   38376 44.099700072
v14          0  0.000000000       4  0.004596592
v15      11457 41.967032967   38379 44.103147516
v16      11471 42.018315018   38424 44.154859172
v17      11446 41.926739927   38350 44.069822227
v18      11456 41.963369963   38376 44.099700072
v19      11459 41.974358974   38384 44.108893256
v20      11457 41.967032967   38383 44.107744108
v21         76  0.278388278     535  0.614794130
v22         90  0.329670330     410  0.471150642
v23      11644 42.652014652   39031 44.852391951
```

3 Shrinkage Methods for reducing dimensions

3.1 Lasso

3.1.1 Theory

In the regression setting, the standard linear model $Y = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p$ is commonly used to describe the relationship between a response Y and a set of variables $X_1, X_2, ..., X_p$. We typically fit this model using least squares. The two best-known techniques for shrinking the regression coefficients towards zero are ridge regression and the lasso.

Ridge regression is very similar to least squares, except that the coefficients ridge are estimated by minimizing a slightly different quantity. We know that the residual square of sums is given by:
$RSS = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}(\beta_j x_{ij}))^2$

In particular, the regression ridge regression coefficient estimates $\hat{\beta^R}$ are the values that minimize:
$RSS = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}(\beta_j x_{ij}))^2 + \lambda \sum_{j=1}^{p}(\beta_j)^2$
where $\lambda \geq 0$ is a tuning parameter. The first term is RSS and second term is shrinkage penalty. The tuning parameter λ serves to control the relative impact of these two terms on the regression coefficient estimates. When $\lambda = 0$, the penalty term has no effect, and ridge regression will produce results same as least squares estimates. However, as $\lambda \to \infty$, the impact of the shrinkage penalty grows, and the ridge regression coefficient estimates will approach zero. Unlike least squares, which generates only one set of coefficient estimates, ridge regression will produce a different set of coefficient estimates $\hat{\beta_\lambda^R}$, for each value of λ. Ridge regression's advantage over least squares is rooted in the bias-variance trade-off. As λ increases, the flexibility of the ridge regression fit decreases, leading to decreased variance but increased bias.

The lasso relies upon the linear model but uses an alternative fitting procedure for estimating the coefficients $\beta_0, \beta_1, ..., \beta_p$. The new procedure is more restrictive in estimating the coefficients, and sets a number of them to exactly zero. Hence in this sense the lasso is a less flexible approach than linear regression. The lasso is a shrinkage method like ridge, with subtle but important differences.

$\hat{\beta} = arg_\beta min(\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}(\beta_j x_{ij}))^2)\ where \sum_{j=1}^{p}|\beta_j|$

Since the overall magnitude of the coefficients is constrained, important predictors are included in the model, and less important predictors shrink, potentially to zero.

4 Classification Algorithms

4.1 Logistic Regression

4.1.1 Theory

We define the class $Y_i = X_i^T.\beta + \epsilon_i$. Now, we have a binary output variable Y, and we want to model the conditional probability $P(Y|X)$. The most obvious idea is to let p(x) be a linear function of x. Every increment of a component of x would add or subtract so much to the probability. The conceptual problem here is that p must be between 0 and 1, and linear functions are unbounded. The next most obvious idea is to let log p(x) be a linear function of x, so that changing an input variable multiplies the probability by a fixed amount. The problem is that logarithms are unbounded in only one direction, and linear functions are not. The easiest modification of log p which has an unbounded range is the logistic (or logit) transformation $log_{10}(p/(1-p))$. We can make this a linear function of x without fear of nonsensical results.

Hence $log(y_i/(1-y_i)) = \beta_i + \beta^T x$ To minimize the mis-classification rate, we should predict Y = 1 when $p \geq 0.5$ and $Y = 0$ when $p < 0.5$. This means guessing 1 whenever $\beta_0 + x\beta$ is non-negative, and 0 otherwise. So logistic regression gives us a linear classifier.

Likelihood of Logistic Function We can essentially see that to find the appropriate $\hat{\beta}$ we need to find the maximum likelihood of the logistic regression function.
$L(\beta_0, \beta) = \sum_{i=1}^{n}((y_i log(p(xi))) + (1-y_i)log(1-p(xi)))$
We can simplify the likelihood function as
$\sum_{i=1}^{n}(-log 1 + e^{\beta_0} + x_i\beta + \sum_{i=1}^{n}(y_i(\beta_0 + x_i\beta)$
Typically, to find the maximum likelihood estimates we'd differentiate the log likelihood with respect to the parameters, set the derivatives equal to zero, and solve. We can solve the derivative numerically in R to obtain the necessary values of β

4.1.2 R Implementation

- We first remove the correlated variables as logistic regression assume uncorrelated variables.
- We also removed all the missing variables.
- Then we encoded those categorical variables which had a range of atmost 4 categories. We used one-hot encoding.
- We then normalized all the variables with respect to the corresponding variable norm (sum of squares)
- We then fitted and crossvalidated the logistic regression model using *cv.glm* function
- Using the final logistic regression model, we predicted the probabilities for the entire test dataset, following similar steps.
- We then uploaded the generated probabilities on www.kaggle.com to find the logloss score which was **0.54610**

4.2 SVM

4.2.1 Theory

The support vector machine is a generalization of a simple and intuitive classifier called the maximal margin classifier. Support Vector Machines are based on the concept of decision hyperplanes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. (Classification tasks based on drawing separating lines to distinguish between objects of different class memberships are known as hyperplane classifiers)

Support Vector Machines are supervised learning models that analyse data used for classification. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a **non-probabilistic binary linear classifier**. A non-probabilistic classifier is one which does not assign a degree of probability to the class which it predicts for the specific input sample i.e. we do not gain any insight into the degree of confidence of the classification decision. Formally, a support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

Suppose some given data points each belong to one of two classes, and the goal is to decide which class a new data point will be in. In the case of support vector machines, a data point is viewed as a - p dimensional vector (a list of p numbers), and we want to know whether we can separate such points with a *(p-1)* dimensional hyperplane. This is called a linear classifier. There are many hyperplanes that might classify the data. One reasonable choice as the best hyperplane is the one that represents the largest separation, or margin, between the two classes. So we choose the hyperplane so that the distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the maximum-margin hyperplane and the linear classifier it defines is known as a maximum margin classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. The kernel trick is a transformation of the feature vector from the given features to a new feature which is defined as the similarity of the point with the other training points and the measure of similarity is determined by the type of kernel like Gaussian, polynomial, etc. The model is then trained on these points with transformed feature vectors and the weighting coefficients are determined according to an optimized loss function consisting of the mis-classification cost and the margin of the classifier.

4.3 Ensembling

Ensemble models are becoming extremely popular due to their ability to give high accuracy predictions, especially in data science competitions hosted online. An example of ensemble model could be a majority vote, a weighted vote, averaging, etc. in the case of classification or prediction of probabilities. A simple illustration will help drive the point about why ensemble models are robust and better i.e. low bias and variance. Suppose we are to make a binary classification decision about 5 test examples and we use 3 'weak' classifiers which output the following results:

11100
01101
10011

Assuming that the true classes of these 5 examples were 11111 then we have 3 classifiers with 60% accuracy i.e. each classifier got 3 of the 5 examples right. Taking a majority vote among the 3 models, our final prediction of the 'strong' classifier would be- 11111, which corresponds to 100% accuracy. Although, a 40% jump in accuracy is an exaggeration we can see how a combination of classifiers which are good at correctly classifying different kinds of examples would make a good classifier. Variants of a majority vote would be to give weighted preference to each model i.e. all of the low-weight models should classify an example differently to overturn the classification decision made by a high-weight model. In the current scenario of probability prediction, weighted or unweighted averaging of the results of models tuned with different parameters can be used.

Inspite of all these advantages of better prediction results, ensemble models are hard to interpret and make sense of in terms of real world variables. Modelling is not always about the prediction results but also about gaining insights into the factors or attributes which influence the outcome. Complicated ensemble models don't give us this intelligible insight which can prompt decisive action in industry.

Pros:

- Invariant to feature scaling i.e. need not pay attention to feature normalization.
- Ensemble models learn higher order interactions between the basic features, which adds information to the predictive model making it more accurate
- Easily scalable and optimized implementations available as packages in most programming languages

Cons:

- Offers very little insight into the factors affecting the outcome, which might be of value in real life problems.
- Sometimes the model maybe too complicated and tuning the hyper-parameters to get the best prediction accuracy can be time-consuming.

Now we implement two models which use Ensembling namely:

1. Random Forest
2. Boosting

4.4 Random Forests

4.4.1 Theory

Random Forest is a kind of ensemble learning method. Ensemble learning methods generate many classifiers and aggregate their results. Two well known methods for ensemble learning are **boosting** and **bagging**. Breiman (2001) proposed random forests, which add an additional layer of randomness to bagging. Random Forest is a versatile machine learning method capable of performing both regression and classification tasks. It also undertakes dimensional reduction methods, treats missing values, outlier values and other essential steps of data exploration, and does a fairly good job. It is a type of ensemble learning method, where a group of weak models combine to form a powerful model. Random forests is a substantial modification of bagging that builds a large collection of de-correlated trees, and then averages them.

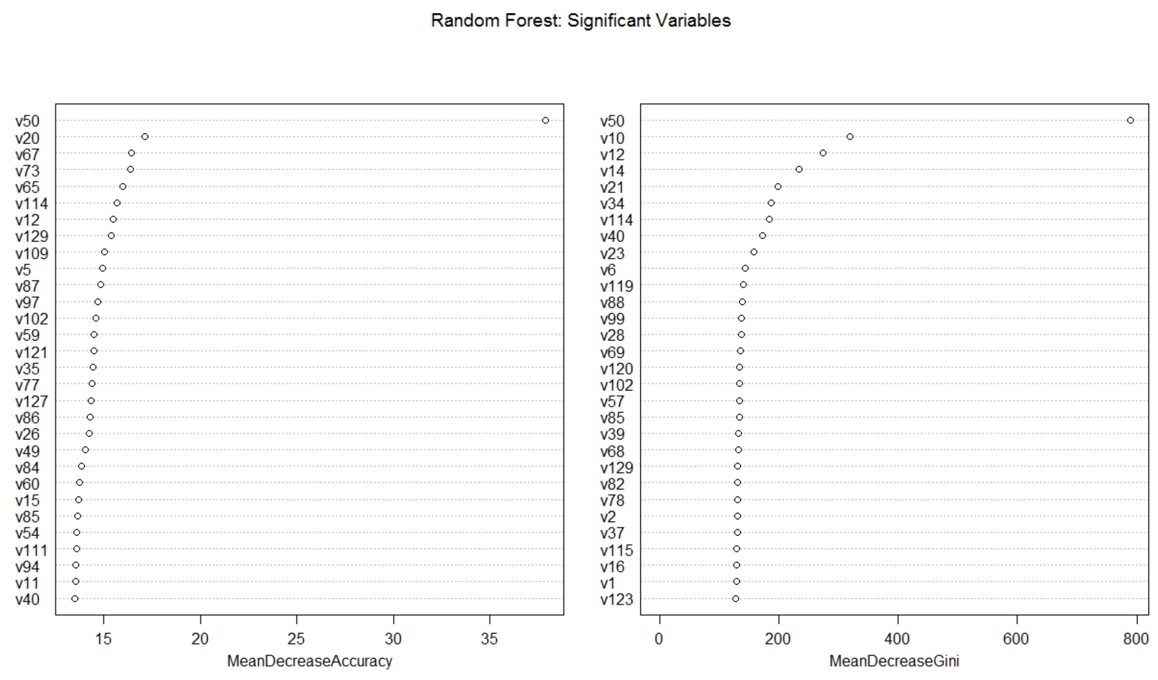
How does a Random Forest work?

- Assume number of cases in the training set is N. Then, sample of these N cases is taken at random but with replacement. This sample will be the training set for growing the tree.
- If there are M input variables, a number m¡M is specified such that at each node, m variables are selected at random out of the M. The best split on these m is used to split the node. The value of m is held constant while we grow the forest.
- Each tree is grown to the largest extent possible and there is no pruning.
- Predict new data by aggregating the predictions of the ntree trees (i.e., majority votes for classification, average for regression)

4.4.2 R Implementation

randomForest implements Breiman's random forest algorithm (based on Breiman and Cutler's original Fortran code) for classification and regression. It can also be used in unsupervised mode for assessing proximities among data points.

- For random forest we have replaced all missing variables which were coded as 'na' to '-1'.
- For all the categorical variables we have ignored those.
- So on the complete training dataset we use the Random Forest Model to train the classifier. This package requires all the variables to be in *'numerical'* format.
- Number of trees to grow: 100. We set this to a higher value so that every input row gets predicted at least a few times.
- We also set the **importance = TRUE** so that the function assesses the importance of each variable.
- After fitting the required model we plot the calculated importance of each variable to see which variables are significant in determining the classifier. The *varImpPlot* is a Dotchart of variable importance as measured by a Random Forest. The following figure illustrates the same.



- In above plot we clearly see that the feature **v50** is very significant. We see the list of features which are significant mentioned here.
- The first graph shows that if a variable is assigned values by random permutation by how much will the MSE increase. Higher the value, higher the variable importance.
- Node purity is measured by Gini Index which is the the difference between RSS before and after the split on that variable.
- A simple way to interpret this plot would be that these variables form a part of prediction power of your Random Forest Model. If we drop the top variable (that is **v50**) from our model, it's prediction power will greatly reduce. On the other hand if you reduce one of the bottom variables, there might not be much impact on prediction power of the model.

We then used our model to predict the posterior probabilities for the given test dataset. We evaluated the calculated probabilities using the Kaggle Evaluation metric (logloss). We got a score of **0.51266**.

4.5 Boosting

4.5.1 Theory

Boosting is yet another approach for improving the predictions resulting from a decision tree. Boosting is a general approach that can be applied to many statistical learning methods for regression or classification. The premise of the theory of boosting is that can we use a number of weak classifiers to build a strong classifier. Here, a weak classifier refers to a model which considers a certain set of features for classification and hence works well while classifying certain examples, but might fail for some test examples. The aim is to cascade a few such weak classifiers which will examine different features of the test point and thus our classification decision is made on a more complex overall. This model is a combination of the 'weaker' models which are easier to tune and more tractable.

4.5.2 R Implementation

'XgBoost' is one of the more popular algorithms which can train accurate classifiers and have been used extensively in competitive contests like kaggle to get the best results. We got a chance to learn about the utility of xgboost in terms of its widespread application and adjusting its parameters to tune it for optimum performance by reading blogs written by kaggle masters and the kaggle forum discussions. This algorithm is widely used in industry today and we will look at the advantages and performance tuning aspects in some detail. **Advantages:**

- Trains models efficiently with all features irrespective of their correlation, unlike logistic or generalised linear models which need uncorrelated features.
- Xgboost can be used to identify the features which are most important in the classification process. These features can then be used to train another model and iteratively get a better model.
- The implementation package has a lot of parameters which can be used effectively to tune the model to correct overfitting or bias. We will talk about the different parameters in detail subsequently.
- The algorithm is extremely robust to feature scaling, outliers, etc. and can be easily used in an ensemble to improve the accuracy of the group of models as a unit.

Cons:

- The algorithm doesn't deal with factor or character variables as features. However, we can convert these variables into numerical variables by one-hot encoding using dummy variables or assign each factor a weight.

Parameters for xgboost algorithm implementation in R and the effect that they have on the fitting of the model i.e. variance of the model:

eta [default=0.3] step size shrinkage used in update to prevent overfitting. After each boosting step, we can directly get the weights of new features. and eta actually shrinks the feature weights to make the boosting process more conservative i.e. lower eta implies reducing overfitting. range: [0,1]
gamma [default=0] minimum loss reduction required to make a further partition on a leaf node of the tree i.e.larger gamma implies more conservative model and less prone to overfitting. range: $[0, \infty]$
max_depth [default=6] maximum depth of a tree influences overfitting i.e. by limiting the max depth of the tree we can reduce overfitting. range: $[1, \infty]$
min_child_weight [default=1] minimum sum of instance weight(hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. In linear regression mode, this simply corresponds to minimum number of instances needed to be in each node. The larger, the more conservative the algorithm will be. range: $[0, \infty]$
There are a number of other parameters which aren't as effective in tuning as the ones mentioned above. These were set to their default value.

5 Conclusion

We compared the performance of the classifier with the following models:

- **Maximum likelihood:** According to the no. of occurences of 1's and the total no. of samples in the training set, we used the maximum likelihood principle to predict the proabability of a sample being 1. This was a very naive approach and gave us the baseline error, so that we could compare future model errors against this to see if we were making an improvement.
- **Logistic regression:** This was the second model we implemented to improve on our maximum likelihood benchmark. We encoded the categorical variables which had maximum 4 unique levels with one-hot encoding and ignored the other categorical variables for this analysis. The feature vectors were normalised and model fitting and cross validation was done using glmnet package.
- **Random Forests:** This is a commonly used technique for classification. It also undertakes dimensional reduction methods, treats missing values, outlier values and other essential steps of data exploration. Using RF, we not only calculated a decent model to fit the testing dataset, but we also found importance of various variables which we would use for analysis. However, this model cannot be used with categorical variables without encoding.
- **Xgboost:** This was a new technique that we learnt and experimented with after seeing its popularity in competitive data contests. Its adaptability to high dimensions, correlated features and robustness made it an efficient model to use for this dataset. Tuning of the model using cross-validation errors and test errors was done by modifying the eta, max_depth, iterations and ensemble parameters.This model gave the best results in terms of accuracy.

To achieve the optimum Log-loss i.e. minimum value of the loss function, the classifier must predict maximum examples in the test set correctly and for those examples that it can't classify correctly it shouldn't classify them to the incorrect class confidently (with high probability). Thus, smoother predictions i.e. values around 0.5 rather than extremes, for points in the test set that can't be classified confidently will give a better Log-loss than classifiers which predict incorrectly with confidence. Hence, a smoothing transformation can be carried out on the final prediction vector to optimize the Log-loss score.

5.1 Results Summary

We implemented the above said classification techniques and uploaded the results on *www.kaggle.com* to check the log-loss scores according to the evaluation metric mentioned in Section 1.3. The results are as follows:

Maximum Likelihood	0.55021
Logistic Regression	0.54610
Random Forest	0.51266
XGBoost	0.45825

As expected XGBoost was the best algorithm followed by Random Forest and Logistic Regression and lastly by Maximum Likelihood Estimation. **Kaggle BNP Paribas Contest**
Team Name: $SSK|MAG$
Current Rank: **980/2554** (Rank 1 has logloss score of 0.42339)
Current Working Code (using XGBoost) is uploaded on Github.
Github links: *https://github.com/miteshgadgil/BNP_mitesh-saurabh*

5.2 Summary of Approach

This project helped us gain exposure to working on a real world dataset with a large number of features and missing data. It was a huge learning curve and we found a great resource like Kaggle forums to help us in our learning in the future. During the course of the quarter, we have done various homeworks and lab sessions which have allowed us to explore the various techniques and tools required in each step of the data analysis pipeline. Following are the key learnings we have obtained while working on the homework assignments, lecture and lab sessions and the final project, about the approach to a data analysis and modelling task:

Exploratory Analysis

- Summarize the variables to look at the range, means and medians of the continuous variables or the unique levels that the categorical variables take.
- Look at the missing value percentage of various attributes and note if any variable does have a percentage greater than 5-10%. Visualize missing data and eliminate data points which have a lot of attributes (75%) features missing. The R-package 'mice' is helpful in working with missing value visualization.
- See if there are noticeable skews in feature values for different target values,by plotting histograms for continuous variables and barplots for categorical variables for all points in the training set.
- Plot correlograms that can help eliminate highly correlated variables thus reducing dimensionality of the problem without loss of information content. A similar analysis for categorical variables can be done by using the "table" command and looking at correspondence between factors.

Imputation and Feature reduction

- Using one of the many packages in R like caret, mice, etc. look at the missing values in features as well as missing data for each data point. Eliminate data points with a lot of missing information and impute missing values of features using the various prediction methods given by the R-package for the continuous variables.In the case of categorical variables, the mode can be used to replace missing factors or a random forest analysis can be done on the categorical variable with the continuous variables as features.
- Datasets with high dimensionality need to be reduced before modelling, since the presence of predictors that aren't influential in classification contribute to the noise in the model which affects the accuracy of prediction. The reduction can be carried out in a number of ways depending on the tools available and the type of modelling technique that we plan to use. We usually start with elimination of highly correlated and duplicated variables. Other useful techniques of feature selection include Lasso for generalised linear modelling, t-statistic significance value for logistic regression and variable importance plots for random forests and xgboost.

Model fitting and Ensembling

- Choosing a model is an important step and is done mostly using experience or domain knowledge of the problem involved. The kind of model also depends on the properties of the predictors i.e. their type, dimension, scale, etc. For purposes of prediction only, where interpretation of the model is not so important, models like random forest and xgboost are used in ensembles. Model interpretation is much easier in case of generalised linear models and Support vector machines.
- Training data is manipulated according to the need of the model and fed to the algorithm so that it returns an object containing the model which can be used to make predictions about the test dataset. Data manipulation includes conversion to a numerical matrix for xgboost, one-hot encoding of categorical variables for logistic regression, etc. and depends on the modelling technique to be used.
- Ensembling is frequently used in combination with boosting models so that predictions from a few "weak" models is averaged to obtain the final prediction which is much more accurate.

Cross validation and Model tuning

- Cross validation is the process of checking if the model fits the training set well for us to be confident to use it on the test set to make predictions. K-fold validation can be easily implemented using the methods built into the R package of the model itself.
- Model tuning is an acquired skill and probably takes the longest time to perfect. It involves numerous iterations of modelling the data over the various combinations of tuning parameters and checking the cross validation error and training error. Based on these 2 errors we can see whether our model is over-fitting and not generalizing well i.e. decreasing training error but increasing testing error or under-fitting i.e. both training error and testing error are increasing. The best model is one which gives the least combined error (bias and variance) also called the mean square error.

Result visualization and analysis

- Usually the feature variables of a dataset are known and hence a model can be used to find out what factors influence the target/outcome the most. Such an analysis can help executives make decisions and decide on a plan of action, while focusing on these influential factors.
- Analysis of the results is best presented to a lay person using visual aids like graphs and plots. Interpreting the results as they apply to the specific problem domain makes it easier for executives to make decisions.

References

[1] Roger Peng *Exploratory Data Analysis in R.* Leanpub, 2015.

[2] Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani *An Introduction to Statistical Learning.* Springer Texts in Statistics, 2013.

[3] Trevor Hastie, Robert Tibshirani, Jerome Friedman *The Elements of Statistical Learning.* Springer Series in Statistics, 2008.

[4] John Maindonald, John Braun *Data Analysis and Graphics using R.* Cambridge University Press, 2003.