Wilfrid Laurier University's Mathematics Department

RISK CLASSIFICATION FOR LIFE INSURANCE

ST 694 : Statistical Learning

By

JAITIKA SINHA (215836410)
DOUGLAS BOWEN (160303920)
RYAN GAUTHIER (140780700)

Table of Contents

EXECUTIVE SUMMARY

Over the years, life Insurance organizations have been endeavoring to sell their products proficiently. And it is hence realized that before an application is acknowledged by the insurance organization, a series of investigative analysis must be embraced during the endorsing procedure.

The introduction of Big Data technology has altered the way an insurance company accumulates, processes, and manages data more efficiently. While companies are carrying out judicious assessments to improve their business adequacy, this increases their tendency to encounter a fraudulent or a High-Risk life in the process. Specialists are now focusing on information mining frameworks to recognize the fakes among insurance firms. Risk profiles of individual candidates are being examined by the life coverage business so that the dangers are assessed, and premiums are priced as precisely as conceivable to continue smooth running of the business. Risk classification is a typical term utilized among insurance agencies, which alludes gathering clients as indicated by their assessed degree of dangers, decided from their recorded information. Failure in distinguishing these risk factors can also create an issue referenced before known as “adverse selection”.

A simple example of adverse selection would be to onboard a medically risky candidate onto the insurance company’s portfolio who then ends up making an insurance death claim soon after the start of their coverage. This could be severely loss making for the insurance company, having received very little premium income.

In the proposed work we will use a dataset that is given by Life Insurance Company and that is further used for analysis purpose so that risk assessment can be done.

During implementation we have done dimensionality reduction so that feature can be reduced. Dimensionality reduction is needed because our dataset contains almost 128 features which is highly dimension. The high dimension dataset is considered to be very complex, so dimensionality reduction is needed. There are two main approaches in dimensional reduction which are feature selection and feature extraction. Our research work covers methods on feature selection. This contributes towards the focus of the report, which is to develop a simplified model for quickly and accurately binning life insurance applicants into risk profiles or classes. Multiple modelling algorithms have been explored towards this purpose.

INTRODUCTION

In 2013-14, US census data, estimated the life expectancy of American females and males at about 81 and 76 years respectively. With population on the rise, an insurance company had a market of roughly 776 billion dollars as of 2015. Because the market was so large, "Prudential" was eager to capture the potential life insurance buyers. However, the barrier to sales was the onerous life insurance application process which required a large amount of time of the applicant, intensive manual labour to convert the policy and of course, huge amount of data. This severely increases the turn around time, at the end of which a prospective client could be lost despite undertaking the cost towards their acquisition.

Prudential published a data set through Kaggle containing 59,381 insurance-applicant observations and nearly 128 predictor variables that may be used in modelling a single, eight-level insurance-risk response. Due to the sensitive individual information, they contain, the variables were normalized and coded with little additional information except their general nature (e.g., medical history, family history, medical keyword, etc.) and their nominal, interval or class-variable status. Normalized height, weight, BMI and age are also in the data set. The method used to normalize the variables is unknown. Thus, it is impossible re-transform the normalized values into their original values.

This data has been used to build our model and provide a solution to the problem which has been clearly laid out in the following page.

PROBLEM STATEMENT

Providing solutions to enhance risk assessment among life insurance firms using predictive analytics. i.e., Identify the risk classification (0 – Low Risk OR 1 – High Risk) of each potential individual interested in purchasing a life insurance policy from Prudential. Making it a **Binary Classification Problem**.

This helps Prudential in quoting accurate premium rates commensurate to the level of risk within a short turnaround time to avoid losing out on the potential buyer.

EXPLORATORY DATA ANALYSIS

Data Snapshot

Our data source comes from an old Kaggle competition "Prudential Life Insurance Assessment". In the initial dataset provided, the goal is to predict the "Risk Classification" of a potential client's life insurance application. To this end, we have an ordinal Response variable taking the form of 1-8 classes that represent risk classes or levels. To predict said classification , we're provided with the following potential predictors:

Variable
ID
Age, Height, Weight, BMI
Product_Info 1-7
Employment Info 1-6
Insured Info 1-6
Insurance History 1-9
Family History 1-5
Medical History 1-41
Medical Keyword 1-48

- All of these variables, aside from unique ID, come normalized and completely anonymized.
- Aside from Age, Height, Weight, BMI, Product Info 4, Employment Info 1/4/6, and Family History 4, values are integers generally ranging from 1-3. In the case of Medical Keyword, these are binary entries used as dummy variables.
- With this information it is our goal to predict the final risk status of an applicant.

Data Processing (Cleaning and Adjustments)

The data given to us came in both a train.csv and test.csv variant, but the test.csv file was quickly discarded due to lack of response given to actually reference with. As such, the train.csv become our full dataset where adjustments were made.

The first step made was simply to remove the unique ID column. Next, we examined columns for the number of missing entries and removed those that surpassed 40% as it would make it hard to impute accurately in this case. The columns with missing values between 0 and 40% were imputed upon with the mean being used for continuous data and mode for categorical data.

Next, some slight feature engineering was done to break one variable, Product Info 2, into two columns, one representing a letter and one a number as the entries matched a letter/number pair. Additionally, the medical keyword dummy variables were summed to provide a clear picture. Another part to this was also grouping factors that appeared at a very low rate into a single category (0.05% or less was grouped to be exact).

A visual analysis of outliers was performed at this stage as well but found nothing significant. We also looked for any potential cases of high correlation that might cause multi-collinearity issues down the line. BMI and Weight were found to be correlated, but as this was the only instance in so many variables with only moderate correlation, it was not removed.

Lastly, we examined the 8 classes for any imbalance and found that class 8 had a large amount of data while some of the other classes had much less. As this response is an ordinal integer, we converted this to a binary problem of “High” and “Low” risk, where High risk is 7/8 and Low is everything else. In this way, we manage to correct for imbalance somewhat while also providing a useful binary metric to examine.

Overall, our original 127 predictors were reduced to 72.

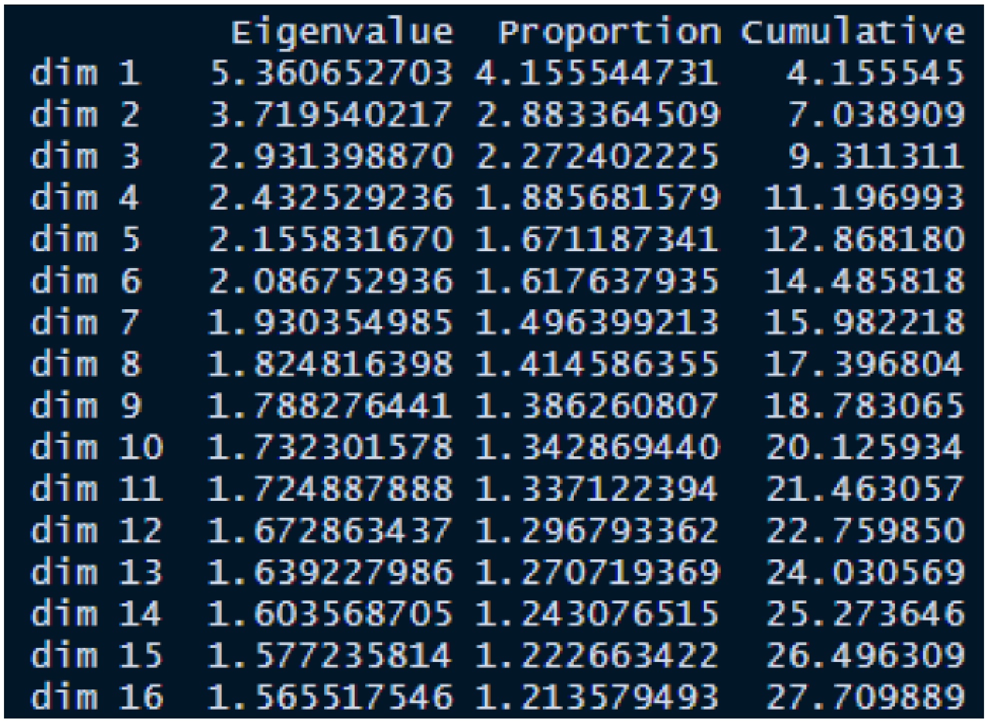
DIMENSION REDUCTION : Principal Component Analysis

In examining our dataset, one of the first considerations made was on whether we could sufficiently reduce the dimensions to make it easier to work with. The go-to method of PCA, unfortunately, is best utilized on continuous variables. It is possible to one-hot encode (mentioned below) the dataset prior to running PCA, but the binary results do not tend to work well as PCA tries to minimize variance.

Still, for the sake of trying, PCA was run with specifications for quantitative/continuous columns as well as qualitative/categorical ones to one-hot encode.

The end results were quite poor as expected, with the first dimension explaining ~4%, the next ~3%, with each subsequent dimension explaining less and less. As a result of these findings, PCA was not utilized.

Snapshot of the first 16 dimensions :



```
        Eigenvalue  Proportion Cumulative
dim 1   5.360652703 4.155544731   4.155545
dim 2   3.719540217 2.883364509   7.038909
dim 3   2.931398870 2.272402225   9.311311
dim 4   2.432529236 1.885681579  11.196993
dim 5   2.155831670 1.671187341  12.868180
dim 6   2.086752936 1.617637935  14.485818
dim 7   1.930354985 1.496399213  15.982218
dim 8   1.824816398 1.414586355  17.396804
dim 9   1.788276441 1.386260807  18.783065
dim 10  1.732301578 1.342869440  20.125934
dim 11  1.724887888 1.337122394  21.463057
dim 12  1.672863437 1.296793362  22.759850
dim 13  1.639227986 1.270719369  24.030569
dim 14  1.603568705 1.243076515  25.273646
dim 15  1.577235814 1.222663422  26.496309
dim 16  1.565517546 1.213579493  27.709889
```

ONE-HOT ENCODING

For the Neural Network & SVM sections, one-hot encoding was used. As a quick explanation, one-hot encoding is a method in which nominal data (i.e., un-order-able categories) is, in a way, standardized to allow different models/algorithms to quantify and map relationships.

Consider for examples sake a column “color” that contains the possible categories {red, blue, green}. For many models, strings are not interpretable. We could convert these to integers {1, 2, 3} respectively - but that might imply some ordered (ordinal) relationship where 2>1. But blue > red is not a quantifiable statement. So simply converting our values to integers is not appropriate. This would be called integer encoding.

We instead move on to one-hot encoding. Instead of converting the categories to integers within the column, we make a column for each category. And so, the “color” column might become “color_red”,“color_blue”,“color_green”. With these columns, we can quickly indicate which color/category an entry falls under using binary {0,1} notation, where a 1 represents true and 0 represents false. In this case only one of the three columns for each row could have a 1 value while the rest would be 0.

As we’ve now mapped these categories to binary values, we’ve avoided suggesting an ordered relationship and instead have made the categories into indicator variables.

One-Hot Encoding Shortcoming: One-hot encoding was performed in **python** through the sckit-learn module where seed generation for splitting test/train sets differs from R. As a result, the train/test datasets for anything utilizing one-hot encoded data will be slightly different due to randomization.

1. LOGISTIC REGRESSION

Logistic Regression is a classification algorithm where the response variable is categorical. Considering, we have a binary classification problem at hand where our response variable is categorical : 0 – Low Risk or 1 – High Risk, logistic regression would most certainly form the foundation of our analysis.

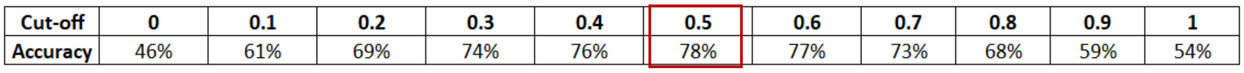
As covered in the previous section, to avoid any data imbalance, the problem was reduced from a multiclass risk rating of 0-8 to a binary rating of 0-1. Furthermore, as noted previously, train data with 72 predictors will be used to fit the model.

Logistic regression in effect estimates the probability of an event happening based on the values of the independent predictor variables or more precisely, the log odds. Resulting values are hence within the probabilistic range of 0-1. These estimated probabilities maximize the likelihood of the response using a binomial model.

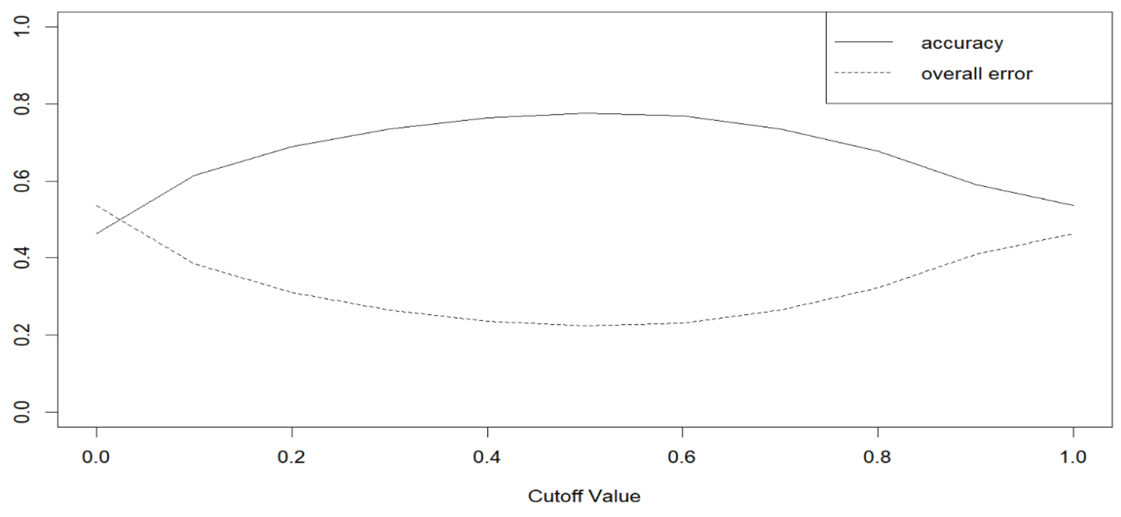
Let's examine results of any 5 observations and how they translate towards a binary output:

			Cut-off = 0.5	**Cut-off = 0.75**
Index	**Actual Classification**	**Model Output**	**Predicted Binary Output**	**Predicted Binary Output**
21	1	0.8886257	1	1
23	0	0.0322061	0	0
28	1	0.8633389	1	1
30	0	0.6329189	1	0
33	1	0.541151	1	0

The estimated probabilities are converted to a binary output depending on the assumed cut-off. The best way to decide what should be the optimal cut-off value, a loop function was determined to calculate the model accuracy at each cut-off value between 0 to 1.



Cut-off	**0**	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
Accuracy	46%	61%	69%	74%	76%	78%	77%	73%	68%	59%	54%



As per observations, a cut-off of 0.5 gives the highest accuracy. This is clearly visible from that graph as well, which peaks at cut-off value 0.5. Assuming this as our cut-off, the confusion matrix looks like the following :

	Actual	
Prediction	**0**	**1**
0	4943	1237
1	1430	4266

Using these values, we can assess our **performance metrics** :

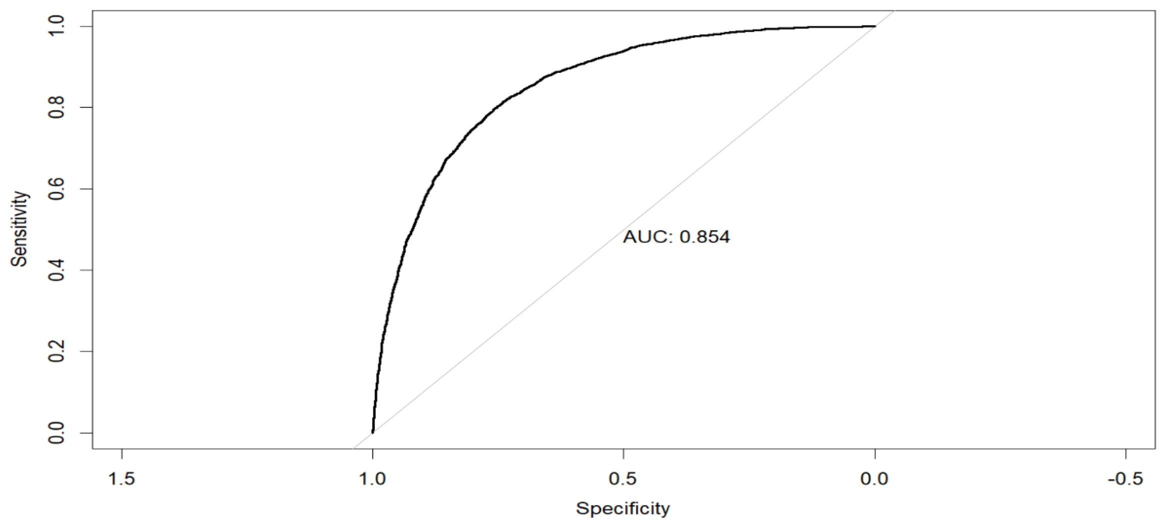
Accuracy	**Precision**	**Recall**	**F1**
77.5%	77.6%	80.0%	78.8%

With an accuracy of 77.5% we have correctly predicted the risk category of several clients. However, a point to be noted is that **recall rate (80%)** is higher than the precision rate (77.6). This works better in our case because we do not wish to undercharge a high-risk client – a false positive is a greater concern than a false negative.

Simply put, in case of a "False Positive" although we should be charging a higher premium due to negative classification, we will end up undercharging because the model predicted them as positive.

Some graphical analysis –

The **AUC** (Area under curve) value is high 0.854 which is a good sign. AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example.



2. RANDOM FOREST

Random forest is a widely used classification algorithm. It is created from the subsets of a data and the final output is based on the majority ranking across those subsets in case of a classification algorithm. It can reasonably be used for regression problems as well where its output is dependent on average of the subset values.

Why would “Random Forest” prove to be a valuable algorithm for classification, more specifically in our problem?

1. It runs efficiently on large data bases. Hence, with our dataset ranging over 50K records it can prove very efficient.
2. It can handle thousands of input variables without variable deletion. 73 input variables being used in our case is on the higher side.
3. It gives estimates of what variables are important in the classification. It’s easy to feel lost when we have too many predictor variables. Random Forest gives us an insight into the key variables when deciding the risk classification. This will answer the question as to what factors must we look at in a customer when deciding to endorse our brand.
4. Random forest algorithm can handle a dataset containing continuous variables and categorical variables which is the requirement in the dataset being used for our model.
5. It has methods for balancing error in class population unbalanced data sets. While the data imbalance was reduced till an extent in our model, it can still pose a challenge seeing how variant the category (response) distribution was before grouping them.

While, the pros certainly outweigh the cons, one major limitation to consider is the run time and memory storage. For large datasets the major memory requirement increases with the number of trees and is the storage required for the large data itself.

We have considered the number of trees (“Ntree”) as 500. This is a large value, enough to give us an accurate set of predictions as the tree would be detailed enough to capture the relevant variables. Additionally, the run time for a large dataset as ours did not seem to cross 15 minutes which was acceptable.

“Mtry” is simply the number of variables randomly sampled at each node. It is to be taken as the square root of the total number of predictor variables (73 in this case). Hence, we consider a value of 9. The value of “nodesize” implicitly sets the depth of the trees. Setting this number larger causes smaller trees to be grown (and thus take less time). Note that the default values are different for classification (1) and regression (5). A “nodesize” of 5 was considered, this would give us a smaller tree than the default value of 1 which would have taken a long time to run considering our large dataset and avoid overfitting. At the same time, underfitting would be avoided if the depth value is not very large, making 5 a reasonable choice.

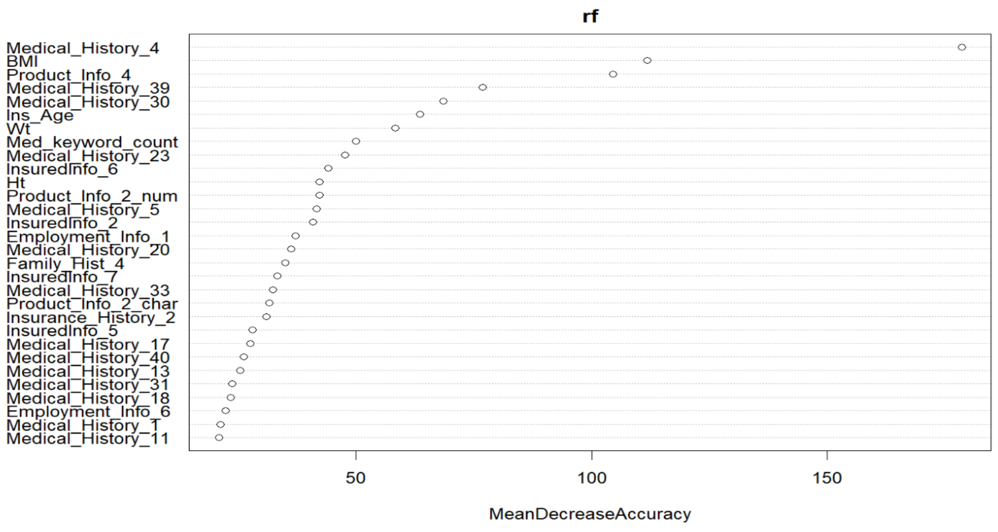
While no hyper tuning was carried out, there were two potential models fit using random forest. The performance metrics of both the models were as follows :

	Accuracy	Precision	Recall	F1
Model 1	80.0%	81.8%	81.0%	81.4%
Model 2	79.9%	81.7%	81.0%	81.3%

In the first model "**Model 1**" we have tried to control the misclassification costs by specifying a loss matrix to weight misclassifications differently. It seemed to marginally improve the overall accuracy and the precision. This feature is mainly a part of the "rpart" package in the decision trees but was implemented here to check if it has any impact. There is another way to explore this but has not been explored in this paper, namely Bagging.

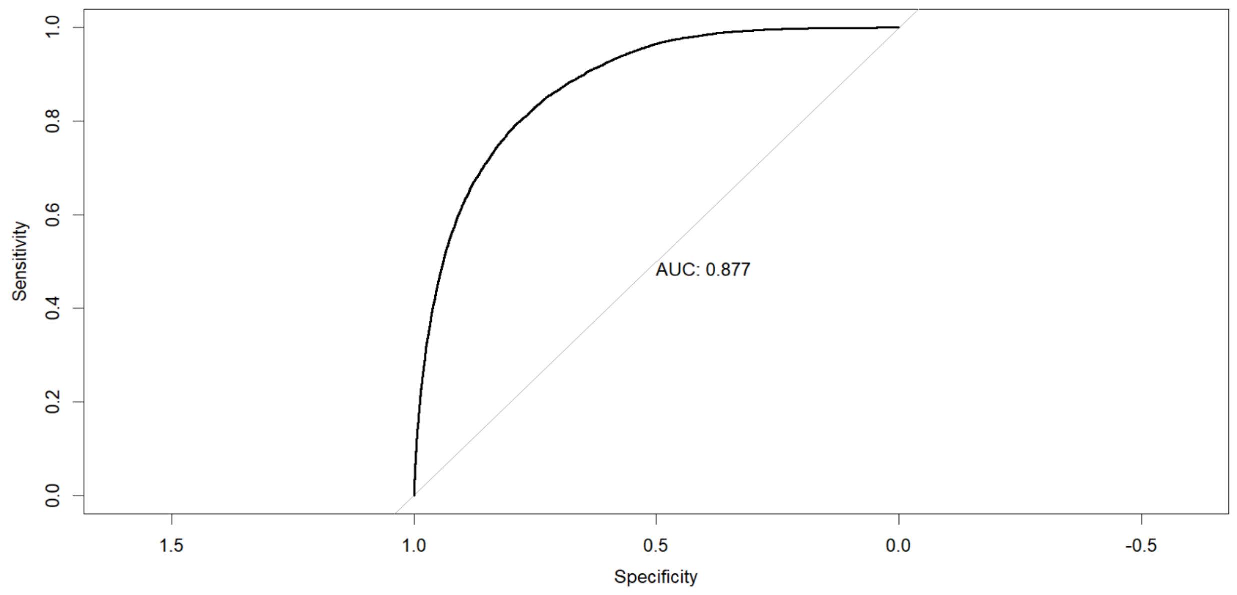
"**Model 2**" is the same as "Model 1", except that we have not allowed for the loss matrix. The results are hence very similar.

Random forest, very clearly aims to answer which variables hold the most importance while making predictions –



Variables such as "Medical_History_4", "BMI", "Product_Info_4" seem to have a high variable importance in deciding the risk classification of the client. Since the data is coded, it is difficult to translate what these variables mean exactly. However, maintaining a healthy BMI and having a good medical history when it comes to say, terminal illnesses will certainly lead to a lower risk factor and hence, a lower premium quotation by Prudential.

AUC Curve estimated on the train data which was used to fit the RF model demonstrated a high AUC value of 0.877. Concluding that the RF model tends to have a superior performance relative to other models for dealing with classification problems such as ours.



3. DECISION TREE

A decision tree is a type of a classification model which is used to making predictions based on how the previous set of questions were answered. Like Random forests, this too can be used to solve regression problems as well. In Decision Trees, for predicting a class label for a record we initiate from the root of the tree. Then compare the values of the root attribute with the record's attribute. Based on comparison, we follow the branch corresponding to that value and jump to the next node.

Decision trees use multiple statistical algorithms to split a node into two or more. This is usually based on the target variable. The creation of the next node increases the desired homogeneity of the sub-nodes. Simply put, the split is such that the purity between the two classes is more evident. The decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes and goes ahead with that one.

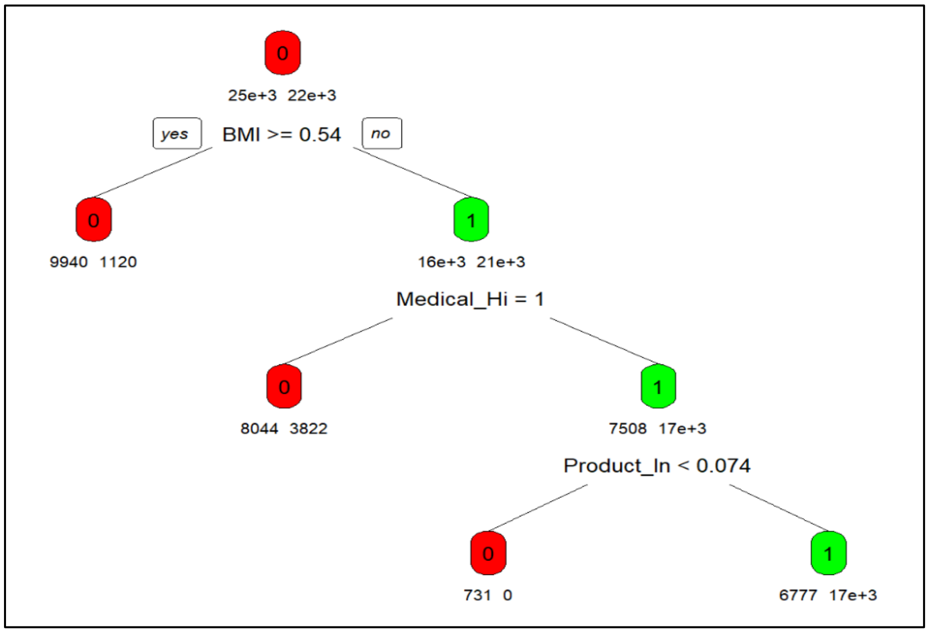
Some key assumptions which are made regarding a "Decision Tree" :

1. At the time of initialising the tree, the whole training dataset is considered as a root.
2. Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model. This suits our datasets wherein most of the features are already of categorical format.
3. On each iteration of the algorithm, it calculates the Gini Index, Entropy(H) or Information gain(IG). It then selects the attribute which has the smallest Entropy or Largest Information gain. This would depend on the choice of the metric by the user, but by default it opts for Gini Index as the cost function used to evaluate the splits in the dataset.

There are several potential models fit using the decision tree. The performance metrics of all the models are as follows :

	Accuracy	**Precision**	**Recall**	**F1**
Model 1	75.9%	73.5%	79.9%	76.6%
Model 2	74.4%	76.4%	76.0%	76.2%
Model 3	76.2%	80.2%	76.5%	78.3%
Model 4	66.1%	39.7%	93.3%	55.7%
Model 5	78.7%	80.2%	80.2%	80.2%
Model 6	72.7%	73.4%	75.2%	74.3%
Model 7	78.7%	80.2%	80.2%	80.2%

Model 1 – This is the basic classification tree which assumes the default parameters. This tree has 4 terminal nodes and is seemingly straightforward to interpret as well.



This fits in well with what we witnessed in the previous section in terms of the Variable importance for Random Forest. Here too, there are 3 predictors coming across – BMI, Medical History and Product Info.

Model 2 – This is the full tree. Argument Cp (Complexity parameter) has been set at 0, which is the smallest possible value.

The complexity parameter is used to control the size of the decision tree and hence select an optimal size of the tree. In simple words this means that the tree construction will not continue into further splits unless it would decrease the overall lack of fit by the value of Cp. Hence, if Cp is 0 then it will continue to split without any constraint. This is why I termed it as "the full tree".

This can lead to overfitting the data which is quite likely as our performance metrics have worsened, compared to the first model.

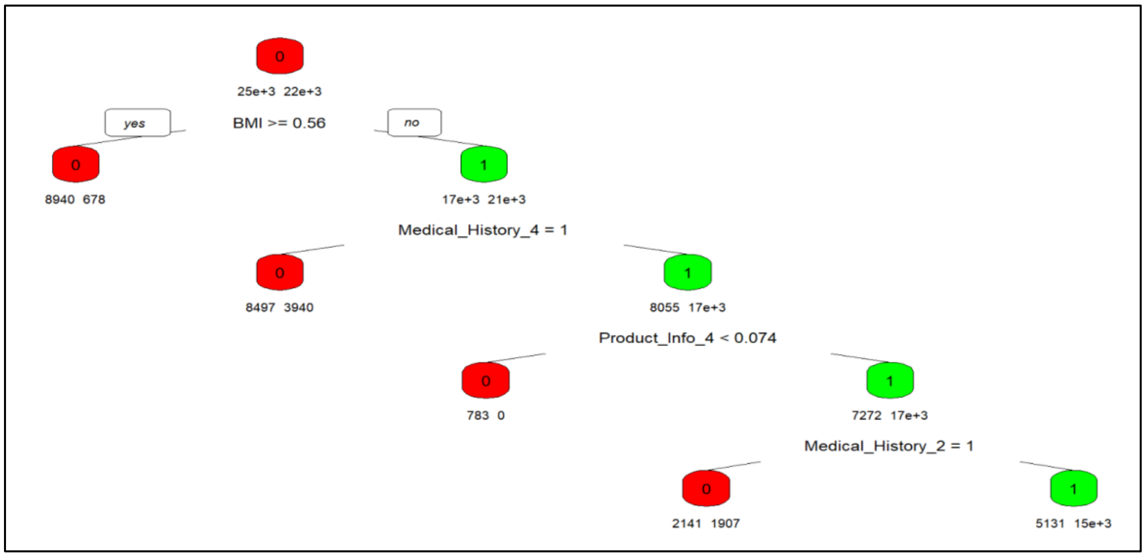
This is what the full tree looks like. Count of the number of leaves (Terminal Nodes) in this case is 1051. This is a huge increase from the 4 in the previous model.

Since the tree diagram would be a messy one it's difficult to interpret anything from it hence avoided pasting in this report.

Model 3 – By default, the rpart package uses the “Gini Index” to assess the impurity while selecting splits for classification. However, if we were to specify a different metric such as “information gain” under the domain “parms”.

In case of using information gain to carry out the splits, it estimates the entropy difference before and after the split to depict the impurity in the class. It aims to reduce the entropy and increasing the impurity while splitting into the next nodes.

As witnessed in the tabulated results, Model 3 has the best performance thus far. It has 5 nodes, and fairly straightforward to interpret like Model 1 but with an improved performance :

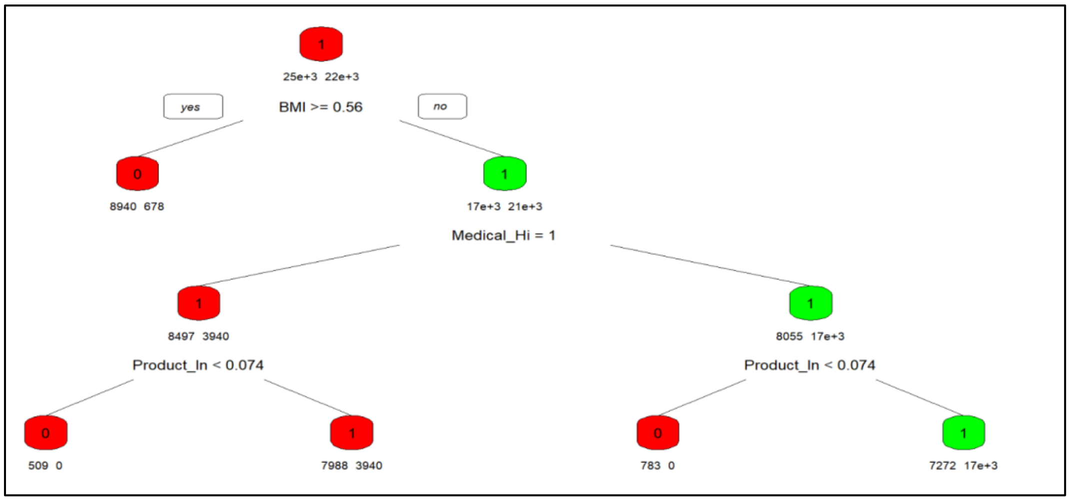


Model 4 – Tried this approach in Random Forest as well. We try to control the misclassification costs by specifying a loss matrix to weight misclassifications differently.

This essentially means that if we were to penalize the miss-classified negative entries more than the miss-classified positive entries. This is a very important model for us because Prudential would want to avoid charging a lower premium to the high-risk customers who are incorrectly classified as low risk.

As we can clearly witness the model accuracy drops significantly because of the fall in precision. However, there is a significant increase in the recall estimate at 93%. The recall value provides this model with sufficient merit to be considered by Prudential.

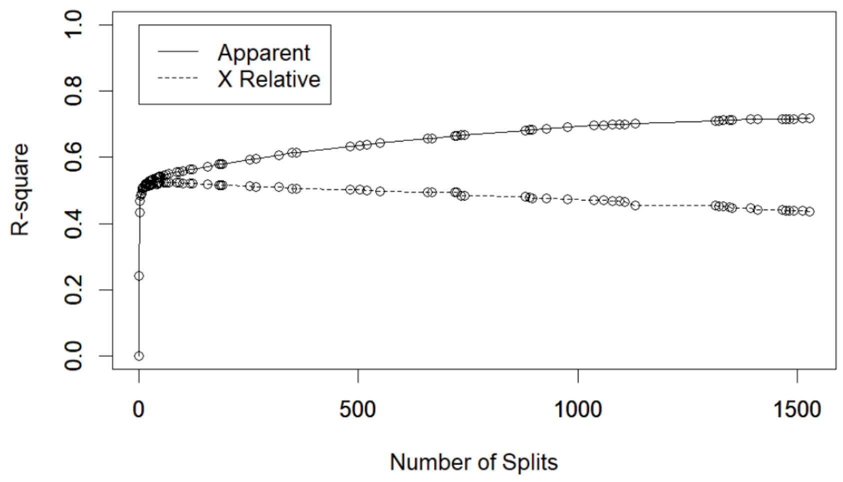
The classification tree has 5 terminal nodes, same as the model above -



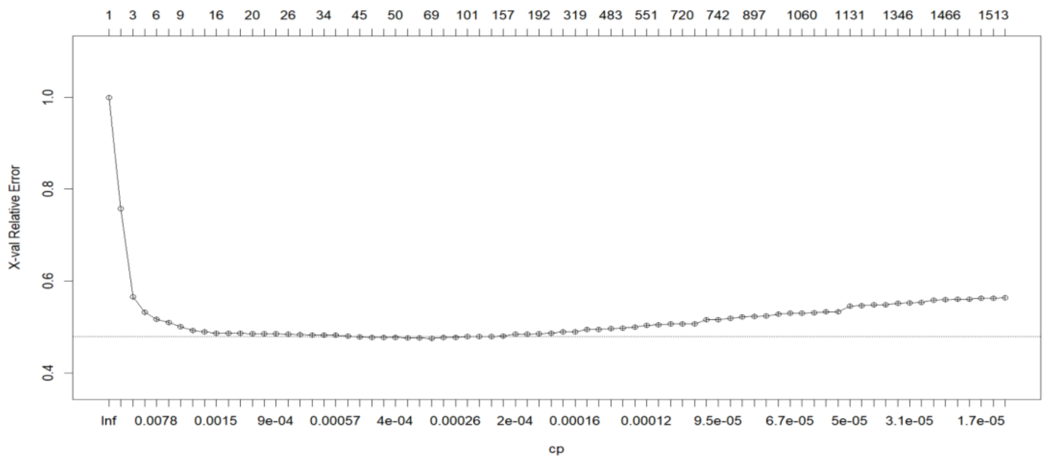
Model 5 – Allowing a constraint on Cp and minsplit argument of the decision tree.
 - Argument Cp has been set at a small value of 0.00001. This means that the split must decrease the overall lack of fit by 0.00001 before attempting to split further. Since this value is low, the tree could be a large one.
- Argument minsplit requires that the minimum number of observations in a node be 5 before attempting to split further. Only 5 observations are required for a large dataset, this will further lead to a large decision tree.

Although the tree is large and messy, it results in a good performance. Best so far. The tree diagram would be a messy one hence avoided pasting in this report.

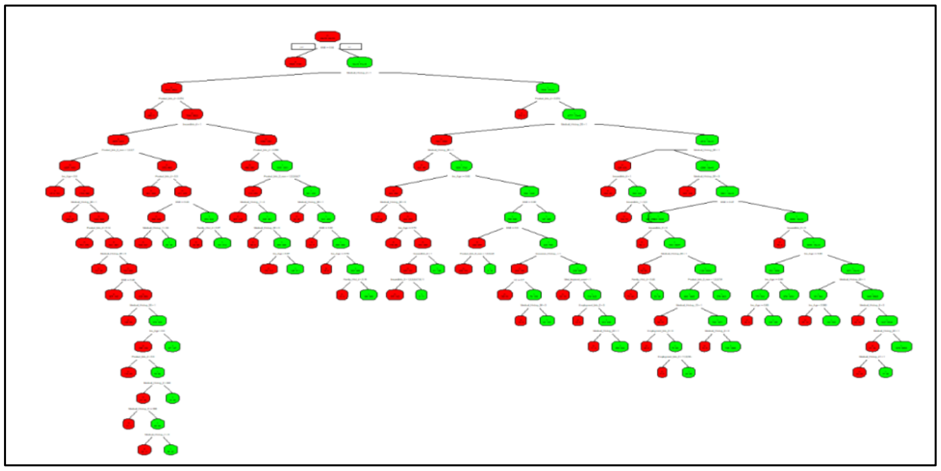
Plot of approximate R-squared and relative error for different split values –



This **graph** shows the fall in standard error if were to split it beyond 3, for a given level of Cp. Hence, helps us in obtaining parameters towards our optimal tree size. -

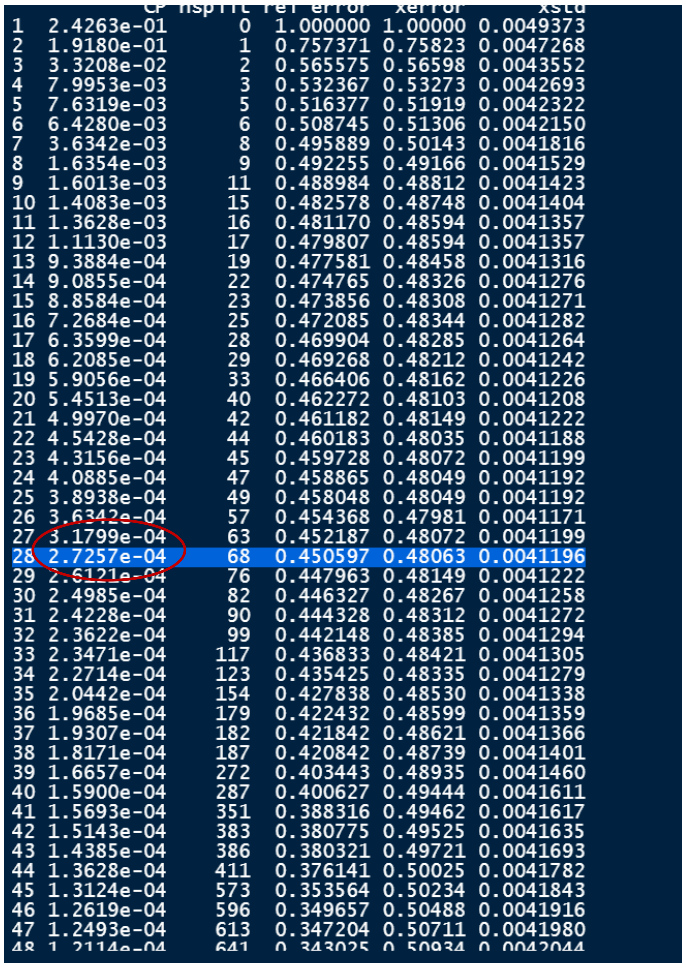


Model 6 – Prune the tree obtained in the previous model which reduces the length of the leaves from 6227 to 69. While this reduces the length of the tree, it also reduces the performance. Keeping this in mind, we move on to the next model which aims at using cross validation = 5 to find the optimal value of Cp such that the standard error is at a minimum.



Model 7 – Now considering a **cross validation** decision tree model.
- Argument "xval" refers to the number of folds to use in rpart's built in cross-validation procedure. This has been set at 5.

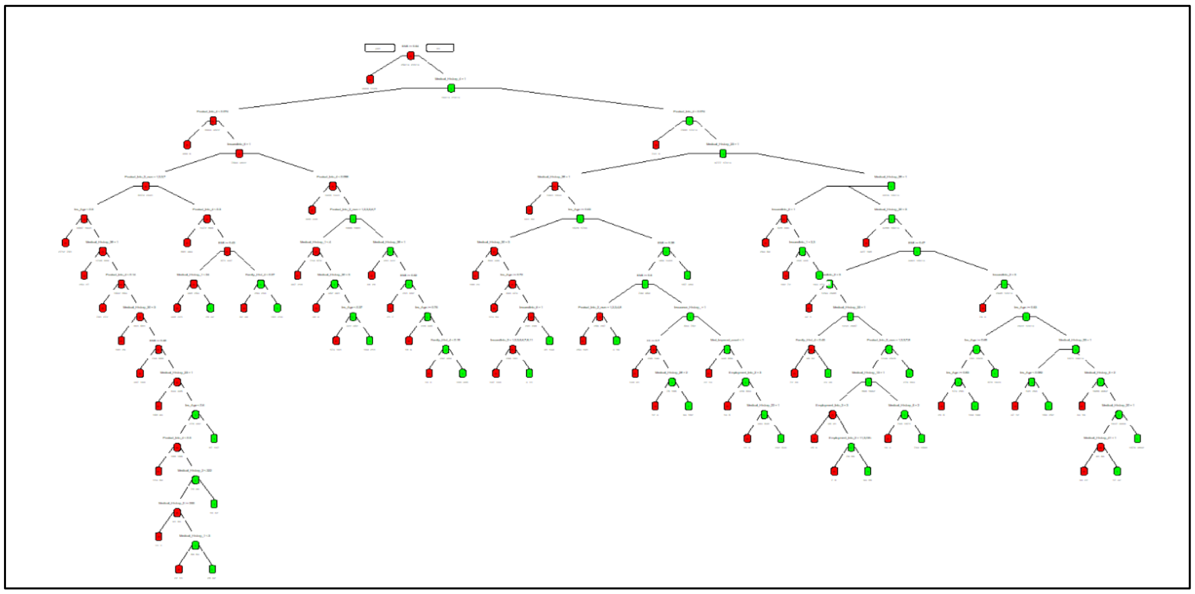
- We first run the model at Cp = 0.00001 and print the following table –



```
          CP nsplit rel error  xerror      xstd
1  2.4263e-01      0  1.000000 1.00000 0.0049373
2  1.9180e-01      1  0.757371 0.75823 0.0047268
3  3.3208e-02      2  0.565575 0.56598 0.0043552
4  7.9953e-03      3  0.532367 0.53273 0.0042693
5  7.6319e-03      5  0.516377 0.51919 0.0042322
6  6.4280e-03      6  0.508745 0.51306 0.0042150
7  3.6342e-03      8  0.495889 0.50143 0.0041816
8  1.6354e-03      9  0.492255 0.49166 0.0041529
9  1.6013e-03     11  0.488984 0.48812 0.0041423
10 1.4083e-03     15  0.482578 0.48748 0.0041404
11 1.3628e-03     16  0.481170 0.48594 0.0041357
12 1.1130e-03     17  0.479807 0.48594 0.0041357
13 9.3884e-04     19  0.477581 0.48458 0.0041316
14 9.0855e-04     22  0.474765 0.48326 0.0041276
15 8.8584e-04     23  0.473856 0.48308 0.0041271
16 7.2684e-04     25  0.472085 0.48344 0.0041282
17 6.3599e-04     28  0.469904 0.48285 0.0041264
18 6.2085e-04     29  0.469268 0.48212 0.0041242
19 5.9056e-04     33  0.466406 0.48162 0.0041226
20 5.4513e-04     40  0.462272 0.48103 0.0041208
21 4.9970e-04     42  0.461182 0.48149 0.0041222
22 4.5428e-04     44  0.460183 0.48035 0.0041188
23 4.3156e-04     45  0.459728 0.48072 0.0041199
24 4.0885e-04     47  0.458865 0.48049 0.0041192
25 3.8938e-04     49  0.458048 0.48049 0.0041192
26 3.6342e-04     57  0.454368 0.47981 0.0041171
27 3.1799e-04     63  0.452187 0.48072 0.0041199
28 2.7257e-04     68  0.450597 0.48063 0.0041196
29 2.6121e-04     76  0.447963 0.48149 0.0041222
30 2.4985e-04     82  0.446327 0.48267 0.0041258
31 2.4228e-04     90  0.444328 0.48312 0.0041272
32 2.3622e-04     99  0.442148 0.48385 0.0041294
33 2.3471e-04    117  0.436833 0.48421 0.0041305
34 2.2714e-04    123  0.435425 0.48335 0.0041279
35 2.0442e-04    154  0.427838 0.48530 0.0041338
36 1.9685e-04    179  0.422432 0.48599 0.0041359
37 1.9307e-04    182  0.421842 0.48621 0.0041366
38 1.8171e-04    187  0.420842 0.48739 0.0041401
39 1.6657e-04    272  0.403443 0.48935 0.0041460
40 1.5900e-04    287  0.400627 0.49444 0.0041611
41 1.5693e-04    351  0.388316 0.49462 0.0041617
42 1.5143e-04    383  0.380775 0.49525 0.0041635
43 1.4385e-04    386  0.380321 0.49721 0.0041693
44 1.3628e-04    411  0.376141 0.50025 0.0041782
45 1.3124e-04    573  0.353564 0.50234 0.0041843
46 1.2619e-04    596  0.349657 0.50488 0.0041916
47 1.2493e-04    613  0.347204 0.50711 0.0041980
48 1.2114e-04    641  0.343025 0.50934 0.0042044
```

The encircled value of Cp gives us the lowest standard error and hence we use that to build our **model 7**. This model by far has the best performance.

The tree diagram for Model 7 looks like this :



4. XGBoost (XGB)

Though not directly covered in this course, at a very high level XGBoost is a decision-tree based algorithm similar to Random Forest but instead of utilizing bagging, it utilizes boosting (stochastic gradient descent boosting to be exact wherein we minimize a loss function). More info on XGB can be found here.

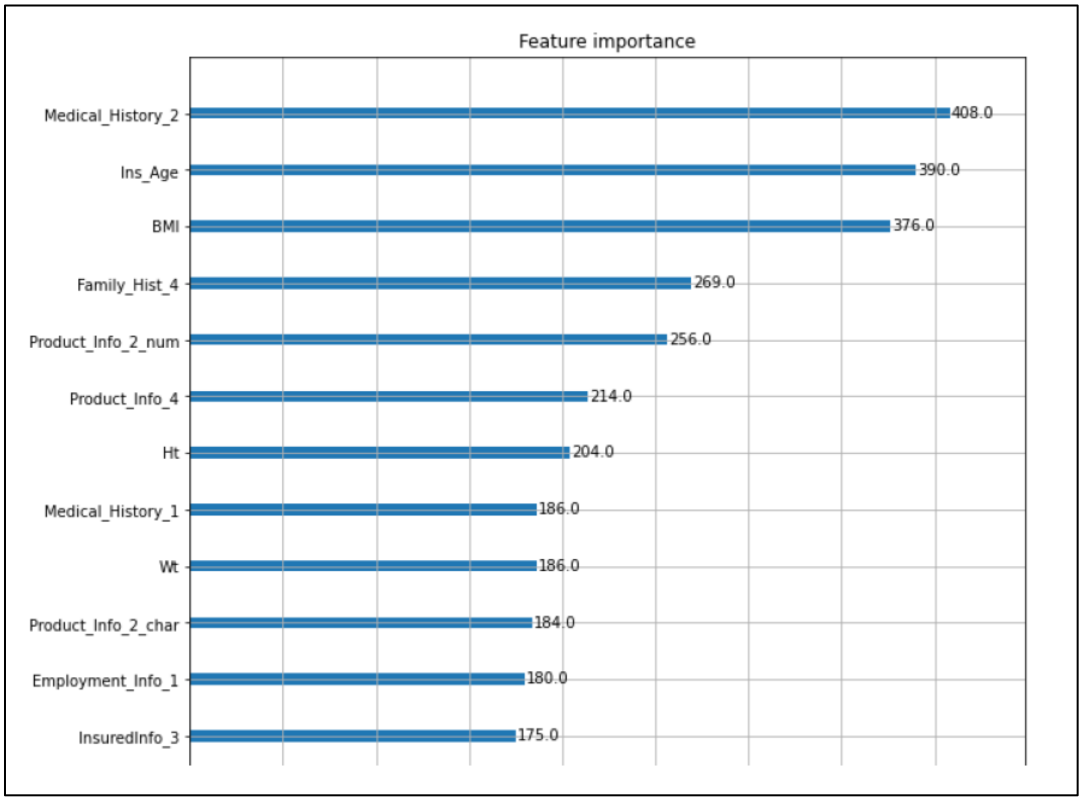
Model Creation

Our XGB model had many parameters that could be set - for our initial model, we chose very conservatively to prevent over-fitting and start as a baseline. We then performed 5-fold cross-validation while examining other parameters, mainly changing those that would make our model less conservative (alpha, eta, depth).

Many variations were attempted, but none provided very significant changes. The largest impact came from changing tree depth, but even depths of 2 with very conservative parameter choices only shifted metrics down by about 3%. All variations had accuracy rates between 76% and 80%, no matter the changes in tree depth, learning rate, and other parameters. Utilizing our best cross-validated model, we ran predictions against the test set. Our metrics were:

Accuracy	Precision	Recall	F1
80.0%	79.0%	77.4%	78.2%

Which is inline with what we've seen for other methods. As a quick examination, we can look at some of the top F-Scores for predictors and see which the model considers to be the most important. After Insured_Info_3, F-Scores drop drastically. A tree diagram could not be included even in appendices due to depth and size but is uploaded to the dropbox under XGB_Tree.png if one wants to examine it.


Feature importance
Medical_History_2
Ins_Age
BMI
Family_Hist_4
Product_Info_2_num
Product_Info_4
Ht
Medical_History_1
Wt
Product_Info_2_char
Employment_Info_1
InsuredInfo_3
408.0
390.0
376.0
269.0
256.0
214.0
204.0
186.0
186.0
184.0
180.0
175.0

5. NAÏVE BAYES

Based on the principal of Bayes theorem, it makes an important assumption of independence between all the predictor variables. This has been implemented because while maintaining its simplicity it is often known to give good classification results for very large datasets as in our case. Although, this will only happen IF the assumption of independence holds true. In real life, it is nearly impossible that we get a set of predictors which are completely independent of each other. Specially in our dataset which is contingent on health attributes.

Taking mere seconds to run, it works well with any computational infrastructure available.

To simply explain the steps which have been carried out in executing this model is –

1. The data is converted into a frequency table which is then translated into probabilities. A dummy example for the categorical field "Product_Info_2" which has 2 factors, has been shown. This approach is replicated for all the variables.

Frequency Table to Likelihood Table

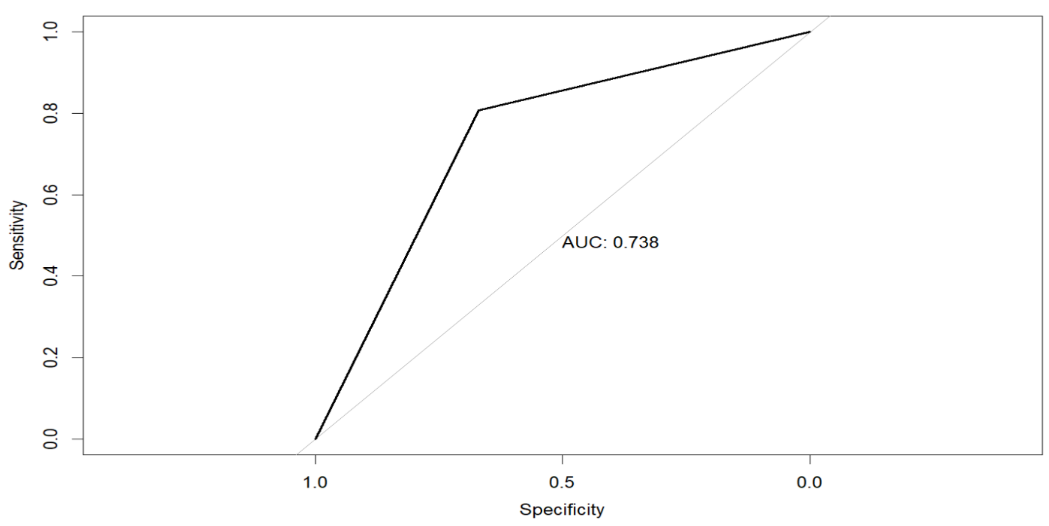
Product_Info_5	**0**	**1**		
2	10	5	=10/38	=5/38
3	15	8	=15/38	=8/38
Total	=25/38	=13/38		

2. The naïve Bayesian equation is then used to calculate the posterior probability for each class 0 and 1 for a given observation. Whichever class has the higher posterior probability, is the outcome of the prediction for that observation.

While there are both merits and demerits to using this approach. Let's see the results obtained using this model :

Accuracy	**Precision**	**Recall**	**F1**
73.3%	80.1%	67.0%	72.9%

Not only is the accuracy lower than that of the other models, but a major concern is also that the Recall estimate is very low at 67%. Considering how a false negative can be loss making for the company, this is a determent towards using this modelling approach.



AUC is 73.8 which seems to be in line with the overall accuracy of the model.

6. K NEAREST NEIGHBOURS

The KNN or k-nearest neighbors' algorithm according to me is one of the simplest machine learning algorithms and is an example of instance-based learning, where new data are classified based on stored and labeled data points.

The predictions made through this approach are entirely data driven and not model driven. It even steers clear of making any data related assumptions.

When predicting the class of any new instance, the distance is calculated between that new instance (Test data point) and the old-stored instance (Train data point). While there are multiple ways of calculating the distance, we have used "Euclidean" to do the same.

Example. If K=3, the distance is calculated between the new instance and the 3 closest data points (Train set). Dependent on the given classification of those 3 data points, the majority class is chosen as a prediction. 2 out of those 3 may be classified as 0 in the train set, in that case even the new instance will be classified as a 0.

What value of K should be chosen?
Few thumb rules to remember – K value should be Odd and not too small or large. Since the value of K is non-parametric, choosing a value K = sqrt(N)/2 is now a general rule. Where N is the number of observations in the dataset. Keeping the above in mind, we have calculated the performance estimates for various values of K. While this would not be an exhaustive list, a loop has been coded to allow for each value of K to be tested. However, this loop has a major drawback in terms of the time and computational infra it needs to run the KNN model for each value of K in a large dataset such as ours.

Before we analyze the modelled results, the data requirements to be satisfied for KNN are :

1. Since we have several factor type variables, we use the "unclass" function to convert the data into a numeric form.
2. Since the data has been modified, we normalize the data again. Approach for normalization has been to subtract the minimum value and divide it by the range.

Once the above changes have been made to the base data, it is then split into train and test to validate out model. Results are as follows :

K	Accuracy	Precision	Recall	F1
1	64.1%	68.0%	63.4%	65.6%
5	68.3%	72.9%	65.7%	69.1%
10	68.8%	74.2%	65.0%	69.3%
20	70.3%	76.6%	65.0%	70.3%
100	69.9%	77.9%	62.0%	69.1%
200	69.1%	78.0%	59.6%	67.6%
250	68.9%	78.0%	59.1%	67.3%

K = 20 seems to offer the best results in terms of the performance metrics calculated at other intervals of K. However, the estimated accuracy is not very high at 70%, with an even dismal Recall of 65%. This can be attributed to KNN being a lazy process where It doesn't allow for a training process to optimize the results. However, it tends to provide quick and simplified results.

7. Support Vector Machine (SVM)

In Support Vector Machine (SVM) Learning, the model attempts to maximize hinge loss - in other words, maximize the distance of a hyper-plane between the two classifications. Another way to think of this is that the SVM algorithm examines only the points that are hardest to classify (the “hinge” cases) into one category or another and then draws a hyper-plane in a way that the cases on one side vs. another are maximizing the margin around the hyper-plane.

The goal of SVM is overall similar to LDA, however, where LDA tries to maximize separability between classes in general for all data points, SVM tries to maximize separability of just the “hard-to-classify” points.

For SVM’s, there are two key points to keep in mind when applying to data:

(1) Data must be normalized so that small differences between dimensions are not exaggerated (e.g., one dimension 100-1000 while another is 0-1).

(2) Ordinal and Nominal Data must be distinguished and handled. As discussed above, in the case of ordinal data we simply convert to integers and standardize. In the case of nominal data, we one-hot encode.

With the explanation done, we now examine the same training data as the above methods, but with nominal categorical columns one-hot encoded. **It should be noted that the parameter tuning was done against a 5-fold cross-validation set while the evaluation metrics are from a testing set (as tuning was very involved, the respective validation metrics are not shown here).**

Linear Kernel

We first examine the SVM algorithm utilizing a simple linear kernel as this can often provide decent results while significantly reducing computing costs. Our evaluation metrics are:

Accuracy	Precision	Recall	F1
77.7%	75.5%	76.3%	75.9%

Considering our results gathered with other methods, a 77% accuracy for a simple linear model is quite good. But maybe we can improve the accuracy by adjusting our SVM Kernel.

Polynomial Kernel

We now examine the SVM algorithm utilizing a polynomial kernel with degree 3, and similarly find:

Accuracy	Precision	Recall	F1
76.6%	74.2%	75.1%	74.7%

which has provided no real substantial change other than to slightly decrease our metrics. A degree of 2 and other higher degrees were also examined yet yielded no substantial change.

RBF Kernel

When utilizing the Radial Basis Function (RBF), it is important to correctly set our C and Gamma values. Our C value indicates the trade off of trying to classify every example correctly or not. A high C tries to classify everything correctly, while a low C allows for some misclassifications. On the other hand, Gamma determines the influence of a single training example wherein a larger gamma, the closer other training examples need to be to be impacted. With the simple default parameters (as in the prior models), we find:

Accuracy	Precision	Recall	F1
75.4%	73.1%	73.6%	73.3%

which again is quite similar yet requires more costly computation.

Proper Choice of C and Gamma:

Due to computational cost, the optimal C and Gamma were not rigorously examined. However, the approach to optimize C and Gamma is still worth mentioning. Simply put, one would create a list for C and Gamma where each value in the respective lists is an exponential increase from the prior entry. The unique matches for these parameters would then be examined to find the optimal values which would serve as a starting point for further tuning.

The default C value is 1, while the default gamma value depends on the fitted data. For RBF, this was 0.015. Though not shown above, and though not a rigorous computational examination, differing C values were tested at user discretion. Our linear model was also tested with C = {3,7}. Findings showed that whether C was 3 or 7, the evaluation metrics were at most 0.5% greater than when C=1.

However, in the case of RBF, changing our default C to 3 took our ~75% accuracy up to ~78%. A further change in C to 7 ended up over-fitting our model (though still higher than C=1). Seeing significant gain with C=3 but some loss with C=7, C={4,5} were also tested. In the end it was found that C=3 was optimal for RBF, giving the evaluation metrics:

Accuracy	Precision	Recall	F1
78.3%	76.5%	76.3%	76.4%

Now in adjusting gamma, we tried various values around the default 0.015 such as 0.01 and 0.02 among others. However, adjustments in both directions yielded no significant increase in metrics.

SVM Analysis Shortcoming:

SVM is an incredibly intensive algorithm with computational cost O($n$features _ n3 observations). As a result of this, the algorithm was only run on smaller subsets of the testing/training datasets so as to avoid incredibly large run-times. As even this subset is still quite large (~10,000 rows by 777 columns post one-hot encode), we believe the results should still be sufficient.

8. Neural Networks (NN)

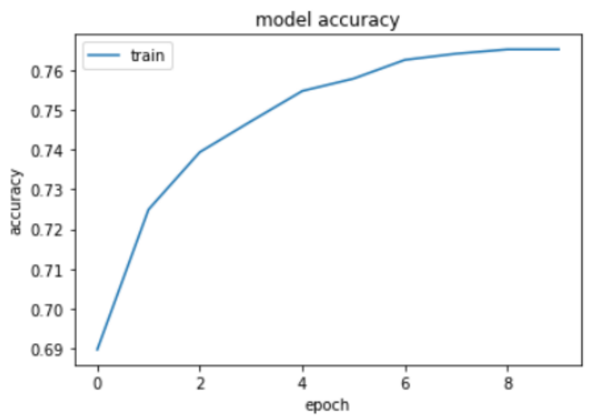
Binary Classification:

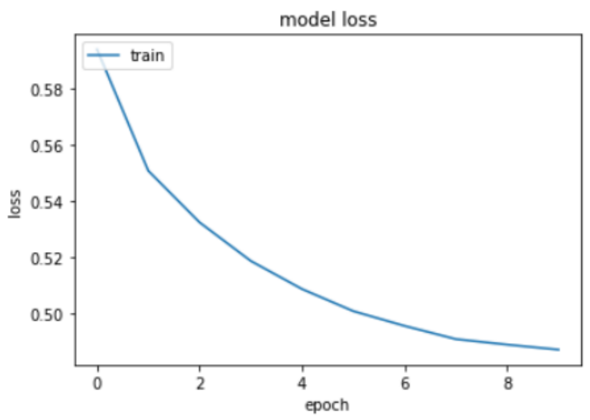
Lastly, we examine a classification attempt utilizing a Neural Network.

We utilize the one-hot encoded data version we mentioned earlier, which results in 777 predictors. The key choices in our network that impact accuracy are:
(1) # of Hidden Layers
(2) Structure of Hidden Layers
(3) Loss Optimizer Function
(4) Activation Function
(5) Epochs/Batch Size

Generally speaking, there is no right choice for the structure of our Neural Network. A large amount comes down to trial and error, especially when regarding binary data. As for loss optimizer, we select binary cross-entropy. We also choose activation function Sigmoid for the same reason. Our Epochs are the number of training cycles and batch size is the number of samples in the training cycle.

We start with a simple network utilizing 1 hidden layer and 8 nodes. We model the accuracy and loss per epoch as follows:





We can clearly see that our accuracy plateaus after epoch 8 at around 77%. This is in line with what we'd expect from other models results. However - keeping only 1 hidden layer and changing the number of nodes, be it 1 or 800 the accuracy stays about the same. What about if we add more hidden layers? We utilize a simple structure of 5 hidden layers with nodes {8, 16, 32, 16, 8} respectively.

Even with more hidden layers and nodes within them, we end up with very similar results by epoch 10. Many more variations were tested and none were able to differ much from 77% by epoch 8.

With this in mind, we will fit a model with 3 hidden layers and nodes {8, 16, 8} and predict with our test data. Our end metrics are:

Accuracy	Precision	Recall	F1
76.95%	77.95%	70.38%	76.97%

Which is again in line with all of our other models.

Multi Classification:

Though our Neural Network performed as expected, we wanted to examine if it had any more success in predicting the 8 response classes pre-dichotomization. As such, we ran similar tests on the same data simply with the response classes in their original states.

Using the same inputs and layer configuration, but now utilizing a categorical cross-entropy loss optimizer, we ran the neural network on the 1-8 classes. The accuracy for this was quite poor at 40%, but when converted back into binary data post-prediction, the accuracy was still low at just 71%.

This is to be expected though as we've now introduced multiple classification possibilities and so the network needs a bit more structure to provide power. We instead try a model with 12 hidden layers and 50 nodes per layer, as well as 32 epochs instead of 10. With this, we found that our accuracy was able to reach nearly 48%, but when converted back to binary, had an accuracy of 76%, which is the same as the binary model produced. It should be noted that with 8 classes the precision was quite poor while recall was high.

Our end metrics post-dichotomization were:

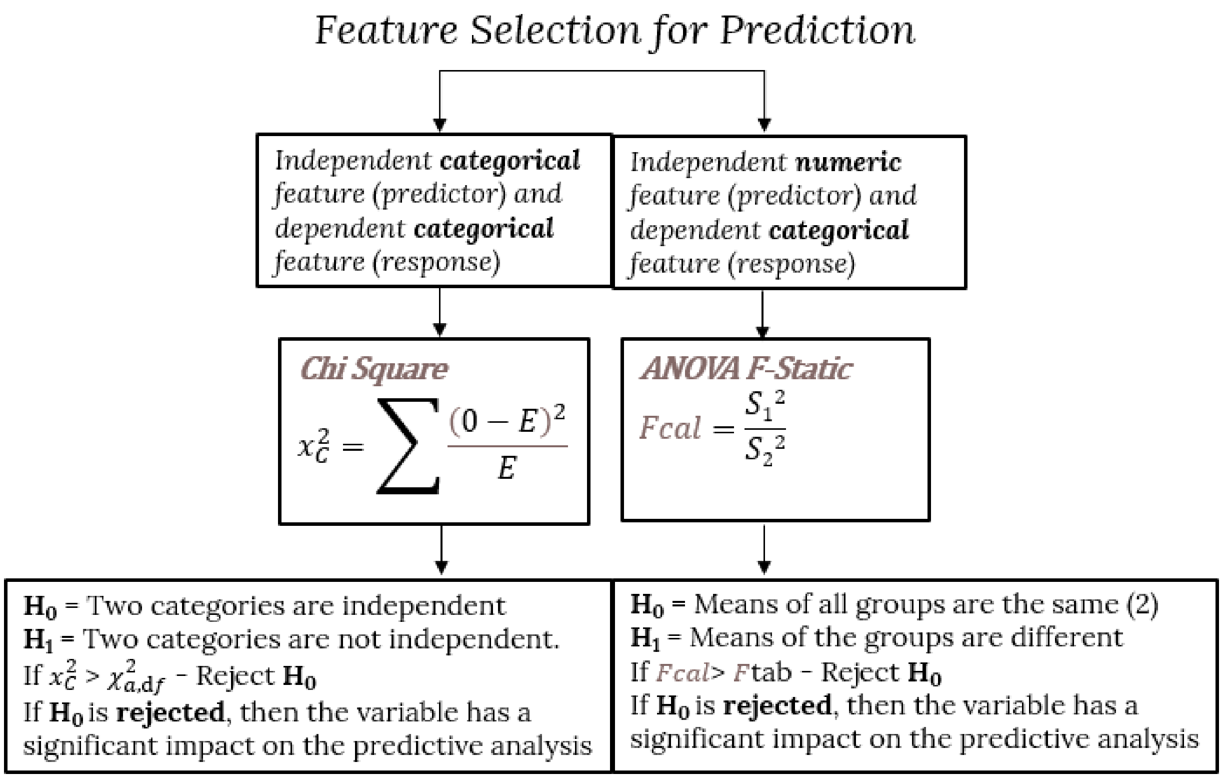
Accuracy	Precision	Recall	F1
76.15%	71.73%	79.95%	75.62%

Overall, neither binary nor multi-class neural network prediction manages to surpass other methods.

9. FEATURE SELECTION : CHI SQUARE DISTRIBUTION AND ANOVA

Chi-Square distribution

In feature selection, we aim to select the features which are highly dependent on the response. When two features are independent, the observed count is close to the expected count, thus we will have smaller Chi-Square value. So high Chi-Square value indicates that the hypothesis of independence is incorrect. In simple words, higher the Chi-Square value the feature is more dependent on the response, and it can be selected for model training.

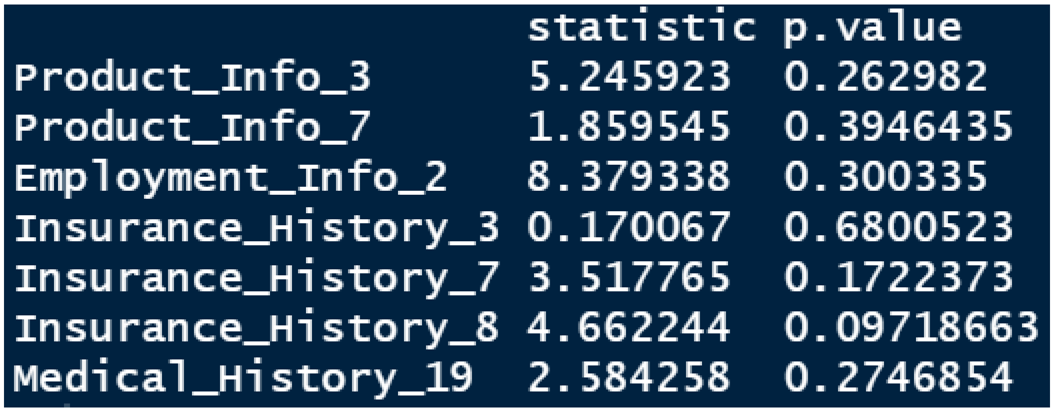

Feature Selection for Prediction
Independent **categorical** feature (predictor) and dependent **categorical** feature (response)
Independent **numeric** feature (predictor) and dependent **categorical** feature (response)
Chi Square
$x_C^2 = \sum \frac{(0-E)^2}{E}$
ANOVA F-Static
$Fcal = \frac{S_1^2}{S_2^2}$
H_0 = Two categories are independent
H_1 = Two categories are not independent.
If $x_C^2 > \chi^2_{a,df}$ - Reject H_0
If H_0 is **rejected**, then the variable has a significant impact on the predictive analysis
H_0 = Means of all groups are the same (2)
H_1 = Means of the groups are different
If $Fcal > F$tab - Reject H_0
If H_0 is **rejected**, then the variable has a significant impact on the predictive analysis

The flow chart explains the statical concept behind feature selection for those features which appear to have a statistically significant relationship to the categorical response variable.

In R studio, we made an attempt to use the function "chisq.test" and calculate the p-value for each of the categorical predictors in relation to the categorical response variable. If the p-value happens to be lower than 0.05, we can reject H_0 and conclude that the two categorical variables are not independent of each other and hence statistically significant.

However, the inverse would hold true if we were to accept H_0 on account of the p-value being higher than 0.05.

The snapshot of R studio, which clearly indicates that the p-value > 0.05 and hence the variables are independent of the categorical response variable. We hence choose to eliminate these variables from our dataset and estimate the performance metrics for our 2 foremost models – Logistic Regression & Random Forest.



```
                    statistic p.value
Product_Info_3      5.245923  0.262982
Product_Info_7      1.859545  0.3946435
Employment_Info_2   8.379338  0.300335
Insurance_History_3 0.170067  0.6800523
Insurance_History_7 3.517765  0.1722373
Insurance_History_8 4.662244  0.09718663
Medical_History_19  2.584258  0.2746854
```

One-Way ANOVA

While there were very few numeric variables, a similar test was conducted for those variables towards assessing their relationship with the categorical response variable. Since, the analysis is now between a numeric and categorical feature, we make use of ANOVA to assess the statistical significance (function aov was used from the package "AICcmodavg").

Here we will check whether there is equal variance between groups of categorical features with respect to the continuous. If there is equal variance between groups, it means this feature has no impact on response and it can not be considered for model training. While the concept remains similar to that shown in the flow chart above, we make use of the p-value to determine the rejection/acceptance of H_0.

It was observed that all the numeric variables had a p-value < 0.05, hence we could not reject the null hypothesis. Resulting in none of features being eliminated from our analysis.

10. IMPACT ON LOGISTIC REGRESSION AFTER FEATURE SELECTION

At the beginning of the modelling process, we had 73 predictors to fit our model. After implementing feature selection, we were able to bring this down to 66 predictors. While the results seem to be fairly consistent, this truly proves that the features eliminated were not contributing to the predictive powers of the model.

With a slightly reduced dataset, the run-time for this model was shaved off by a few minutes (2-4 to be exact).

	Accuracy	**Precision**	**Recall**	**F1**
Before	77.5%	77.6%	80.0%	78.8%
After	77.6%	77.6%	80.0%	78.8%

Barring a slight increase in the overall accuracy, the performance metrics are consistent.

We move on to show the same analysis in case of a Random Forest model.

11. IMPACT ON RANDOM FOREST AFTER FEATURE SELECTION

The results of the new model after feature selection are as follows,

	Accuracy	**Precision**	**Recall**	**F1**
Model 1				
Before	80.0%	81.8%	81.0%	81.4%
After	79.9%	81.9%	80.9%	81.4%
Model 2				
Before	79.9%	81.7%	81.0%	81.3%
After	80.0%	81.8%	81.1%	81.4%

As witnessed in the previous section, the performance metrics are consistent. This clearly goes to show that the feature selection was successful as we were able to reduce the predictors in our dataset without impacting the performance of the model.

12. CLUSTERING ANALYSIS (UNSUPERVISED LEARNING)

The aim towards building this model is to identify any inherent clusters which may emerge from the client's profile (excluding for the response variable). The output resulting from the cluster analysis – Say, Cluster 1 or Cluster 2 is then appended to the train data so that we can use it as a predictive variable.

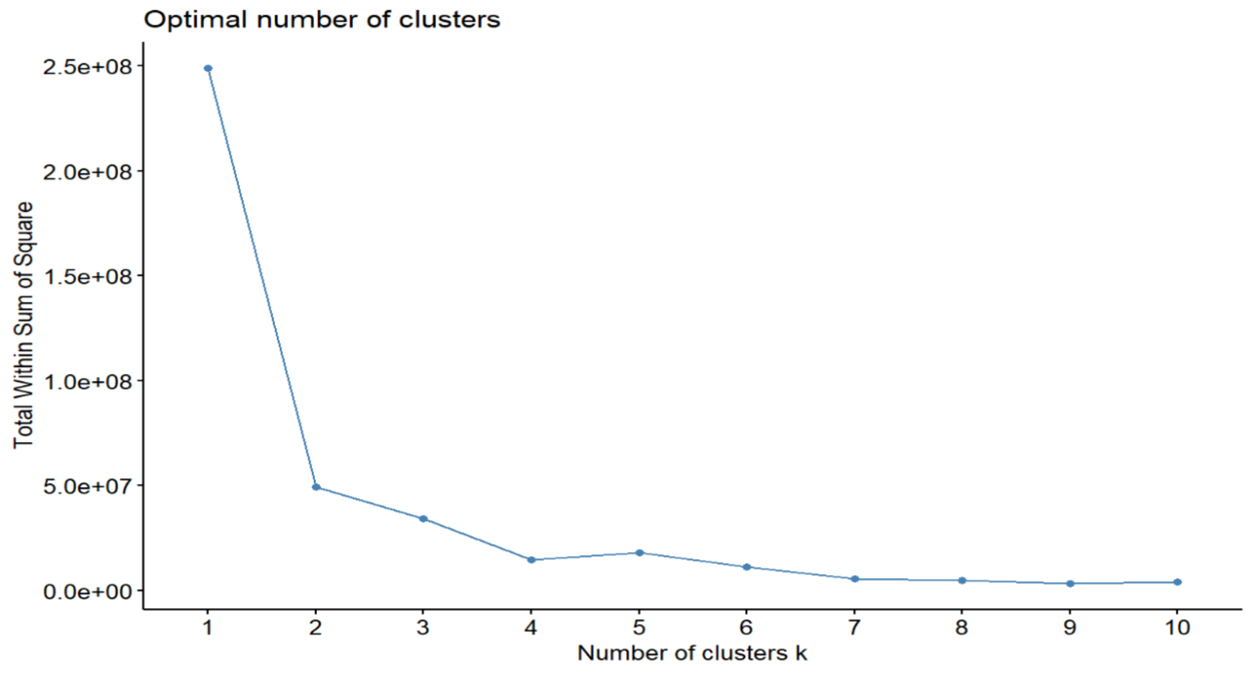
The accuracy of this is then tested by fitting it on a logistic regression model and a random forest model with the extra variable. (Excluding the features which were already filtered due to feature selection).

Few modifications in the data before implementing it in clustering :

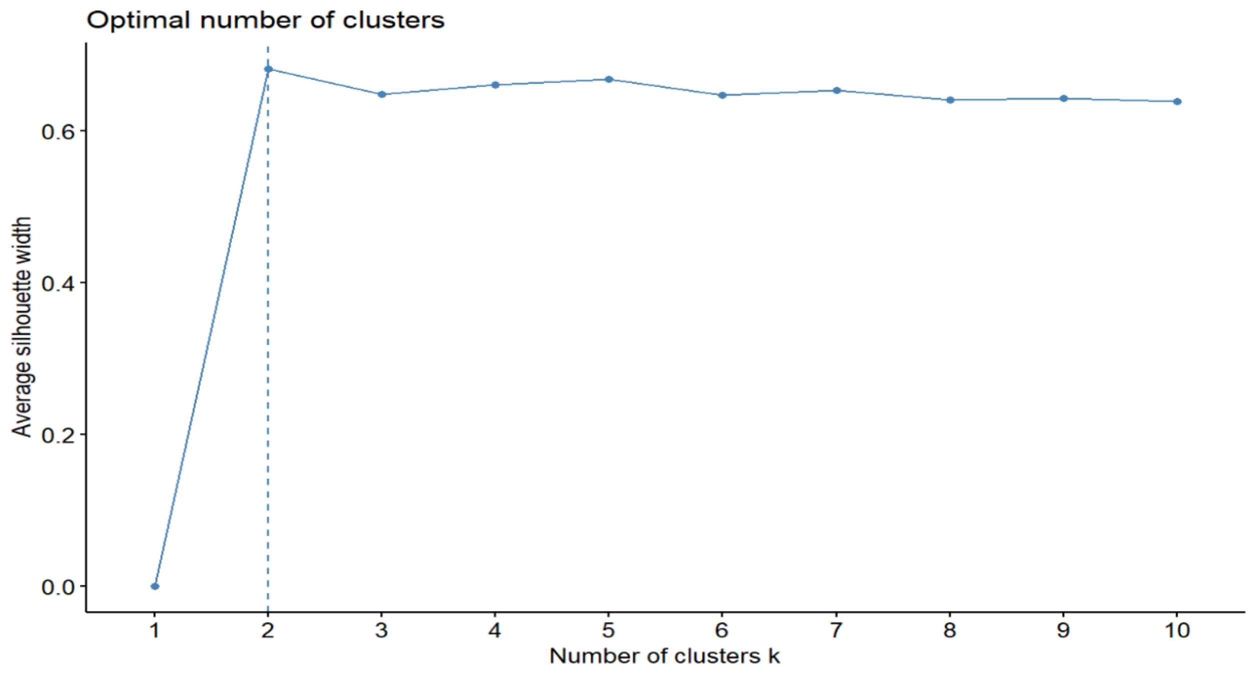
- Since we have several factor type variables, we use the "unclass" function to convert the data into a numeric form.

- A random sample was created as a subset of the training data which consisted only 10,000 records instead of the odd 50,000 records originally. This was done to get some clustering visualisations because the original data was exceeding the size limit.
This is a clear drawback in this process, as it requires a better computational infrastructure and takes time to run if the dataset is large, as in our case.

Plotting an **elbow graph** using the entire dataset, to derive the optimal number of clusters – We witness the elbow curve at 2 and 4. Which could mean that we implement K means clustering with 2 or 4 clusters. Let's further look at the Silhouette method to finalise,



Judging by the graph obtained using the **“silhouette”** method, 2 clusters would be optimal.



13. IMPACT OF CLUSTERING ON SUPERVISED LEARNING MODELS

Using the K means clustering where K = 2, we get an output where each record is either in cluster 1 or cluster 2. This output resulting from clustering, is then used an input while fitting our supervised learning models such as logistic regression, random forest, etc.

But does that have any impact on the predictive abilities of the model? Let's examine,

Impact of clustering on Logistic Regression -

	Accuracy	Precision	Recall	F1
Before	77.5%	77.6%	80.0%	78.8%
After	77.6%	77.6%	80.0%	78.8%
Clustering	77.6%	77.4%	80.2%	78.7%

The performance is nearly the same, which means that the clusters are not adding to the predictive abilities of the model. This means that the client data given to us does not provide definite profile characteristics to solely distinguish the high-risk and low-risk classification.

Impact of clustering on Random Forest –

	Accuracy	Precision	Recall	F1
Model 1				
Before	80.0%	81.8%	81.0%	81.4%
After	79.9%	81.9%	80.9%	81.4%
Clustering	80.2%	82.1%	81.2%	81.6%
Model 2				
Before	79.9%	81.7%	81.0%	81.3%
After	80.0%	81.8%	81.1%	81.4%
Clustering	80.0%	82.1%	80.9%	81.5%

In case of the Random forest, we witness a very minor improvement due to the increased precision. However, the improvement is too slight to credit the clustering in this case.

14. MODEL SUMMARY

Sr. No.	Modelling Algorithm	Accuracy	Precision	Recall	F1
1	**Logistic Regression**	77.5%	77.6%	80.0%	78.8%
2	**Random Forest**	80.0%	81.8%	81.0%	81.4%
3	**Decision Tree**	78.7%	80.2%	80.2%	80.2%
4	**XG Boost**	80.0%	79.0%	77.4%	78.2%
5	**Naïve Bayes**	73.3%	80.1%	67.0%	72.9%
6	**K Nearest Neighbours**	70.3%	76.6%	65.0%	70.3%
7	**Support Vector Machines**	78.3%	76.5%	76.3%	76.4%
8	**Neural Networks**	77.0%	78.0%	70.4%	74.0%

In examining a summary of our "best" models from each method, we find they range from an accuracy of 70% to an accuracy of 80%, a reasonably sized margin between methods. This clearly indicates that there are some models that will perform significantly better with the same inputs for predicting our binary response. However, every model has different degrees of interpretability and especially in a business setting, this is most important.

As such, given that Random Forest and XGBoost are both tree-based and quite easily interpretable in a graphical way as to how the decision was arrived to (it would be quite easy for someone to examine the tree and make a new prediction without a program), we would select Random Forest as our best model as along with being tied for highest accuracy it also has higher precision and recall.

CONCLUSION

Data Analytics is the imperative need of the hour among the organisations around the world. Within the life insurance domain predictive modeling utilizing learning algorithms can provide the eminent contrast with which a business is done as compared with the traditional strategies of the past. Presently, with information arrangements and data lakes, the work should be possibly quicker and with better outcomes. Consequently, it would upgrade the business by enabling quicker administration and service to client. Our model aims to contribute towards this technological advancement by providing meaningful and quicker results which will enable “Prudential” to increase their profit retention and expand their business.

REFERENCES

Data is publicly available at :
https://www.kaggle.com/code/fahadmehfoooz/classification-with-model-interpretation/data

https://towardsdatascience.com/chi-square-test-for-feature-selection-in-machine-learning-206b1f0b8223

https://towardsdatascience.com/anova-for-feature-selection-in-machine-learning-d9305e228476

https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html

Textbook : An Introduction to Statistical Learning: With Applications in R