Vol: 02 Issue: 02

International Journal of Data Science and Advanced Analytics

Machine Learning Application in Car Insurance Direct Marketing

CHENG XiaoTian [1]
[1] School of Computing, Asia Pacific University of Technology & Innovation. Kuala Lumpur, Malaysia
[1] cxtalex@foxmail.com

Corresponding author email: cxtalex@foxmail.com

***Abstract*— Direct marketing such as telemarketing or mailing is an important method for companies to boost their business. Identifying the right proportion of target market could largely cut operational expense and improve efficiency. In this research, a secondary dataset from a car insurance company will be used to study this problem of market targeting. Basing on existing literature study, three classifiers are picked, Naïve Bayesian, Decision Tree, and Neural Network. Some literature researches on each of the algorithm are conducted. Later modelling experiments are performed to predict whether the final customer will purchase the insurance or not.**

***Keywords*— Direct Marketing, Naïve Bayesian, Decision Tree, Neural Network**

1. Introduction

As a critical industry which is closely linked with the welfare of citizens' everyday life, insurance has been given a high emphasis on economy building ever since 40 years ago. Nowadays, there are some major industrial challenges, insurance providers are facing [1]:
Firstly, personalized advice and rapport customer interaction; Secondly, claiming process automation and optimization; Thirdly, detecting fraudulent claims; Fourthly, setting feasible policy premiums; Fifthly, locate high ranked potential customers for direct marketing.
These problems faced by insurers will get a solution with the application of big data and machine learning algorithms [2]. As an intensively data-driven industry, insurance companies could employ the huge amount of data available about their existing customer and enrich that with other dimensions such as social media, transaction record and credit information of the customer, all even mobile data collected from the company website or mobile application. These all help insurance companies building up a multi-dimensional persona of their customer, which is the requisite for providing customized customer service and set up a more fitting insurance premium.
This research is an attempt aiming to find the solution for the fifth challenge listed above. A secondary dataset collected by the Decision Science and Systems Chair of the Technical University of Munich. It is a real-world insurance company who called existing insurance customers up and intend to further sell them auto insurance. The final result whether the customer eventually subscripted car insurance is marked up as "success" or "failure" in the dataset.

2. Materials and Methods

Concerning this problem of identifying potential customers by studying the customer demographic profile, there is a multitude of existing works did by researchers. Table 1 showcases a list of related work did as well as the algorithm they chose and the achieved predict accuracy. The last two are experiments did on the same dataset as this report, and the rest are using similar dataset aiming to find out the relationship between customer demographic profile and the final success rate of direct marketing.
Basing on the algorithm they use for building the classifier, they can be roughly categorized into two groups – traditional statistical techniques, and machine learning algorithms. Statistical techniques are using mathematical formula to describe the relationship between variables, whilst machine learning techniques can learn from data without build up explicit rules and formula [3]. Will pick up three statistical techniques and two machine learning techniques to experiment in this report.

2.1 Traditional Statistical Techniques

2.1.1 Naive Bayesian

The Bayesian theorem is developed in the 18th century by a mathematician called Thomas Bayes, it is a formula for calculating the conditional probability. The theorem provides a method to calculate the probability of an occurring event basing on other events which related to it. Naïve Bayesian is the simple version of it, given the naïve assumption that all the attributes provided are independent of each other. Although this assumption can seldom be fully satisfied, Naïve Bayesian still performs surprisingly well in comparison with other more sophisticated algorithms, especially when the correlation between variables is not that strong. Also, Naïve Bayesian is tested with higher accuracy compared with other algorithms even when dataset size is small.
There are many ways to improve the accuracy of Naïve Bayesian model, for example, perform feature selection to form factor subsets which relatively independent of each other or use Kernelization selection on the attributes which are not normally distributed, or Laplacian Smoothing on the value of the attribute which has zero occurrences in the training dataset.
In the experiment did by [4], and an accuracy of 67.9% is achieved when applying on deciding the target selection of direct marketing.

2.1.2 Decision Tree

The decision tree is a data mining technique to build a predictive model for both classification and regression. The output of the decision tree would be a hierarchical simulation of the flow of decision making. The purpose of the classification decision tree is to classify an instance into predefined classes basing on other related factors/ attributes [5]. In our case, the decision tree is used to define whether the subject customer would like to subscribe to the car insurance basing on its demographic profile.

The process of building the decision tree is a recursive procedure. It will start from a "root" node, then further split into two or more sub-nodes basing on one attribute or more. The sequence for picking up the decision-making attribute is depending on the weight of attribute. The operation will repeat until an utmost homogeneity is reach at the sub-node, which is called as the "leaf". Each path from the root of the tree till the end of its leaf can be interpreted into a rule. Given this, analysts could apply the "tree" with a new instance or dataset, and predict the probability it can be turned into a customer (by sorting the attribute value of the customer down the decision tree), and understand the proportion of potential customer rate out of the given dataset. [5]

The decision tree has its advantages of simplicity and easy to interpret compared with other algorithms. Usually, a classification tree can be depicted in a hierarchical graphic way, however, when the complexity of the classification tree getting too high (too many nodes and high tree depth), this graphical way will not be that helpful. Therefore, in decision tree method, it is necessary to keep the tree pruned as long as the tree can yield the upmost predictive accuracy.

In the experiment [6] built up Decision tree model and Naïve Bayesian model for the predicting of the likelihood of positive response from direct marketing. In their case, the decision tree provided a very good performance of 93.96% of accuracy, while Naïve Bayesian provides the accuracy of 84.91%.

2.1.3 Support Vector Machines (SVM)

Comparing with Decision Tree, SVM is more flexible because no a priori restriction is required in SVM. Besides, comparing with a traditional statistical model such as Linear Regression, SVM could cope with linear as well as complex nonlinear problems. Because of its flexibilities, SVM can produce higher prediction accuracy, but the downside is the output of the model is more complex to interpret for human understanding than Decision Tree [6].

SVM is a binary classifier which can transform an input dataset into a high dimensional data by choosing a kernel and using nonlinear mapping. The SVM then identify the best separating hyperplane which splits the two classes. Before performing SVM modelling, usually, the dataset is standardized and normalized with a mean of zero and a standard deviation of one. There are two mandatory parameters when using SVM, *C* and *gamma*. By changing the value of these two parameters, the model can be tuned with optimized performance. *C* parameter is the trade-off between the smooth decision boundary and classifying training points correctly, while the *gamma* value indicates how far the influence of a single training example reaches. [7]

There are two methods to interpret the result of SVM, one is through rule extraction and the other is by sensitivity analysis. Rule extraction is referring to utilizing white box approaches such as decision tree to understand black-box approaches such as SVM, and the sensitivity analysis works by analysing the output of the model when the input is varying within the domain range.

One study did a comparison study of four different classification models with a dataset collected from a Portuguese retail bank. This is a high-dimensional dataset with 150 attributes related to the client's bank account and social-economic attributes. A feature selection process is performed and cuts down the dataset with only 22 most related features. Then logistic regression, decision tree, neural network and support vector machine are applied on the dataset. Receiver Operating Curve (ROC) and the area of the LIFT cumulative curve (ALIFT) are used as evaluation metrics. SVM is proved with the second-best performance, an accuracy of 76.7% following the best performance model generated by a neural network with an accuracy of 79.4%.

2.2 Machine Learning Algorithms

2.2.1 Neural Network (NN)

Another powerful classifier is NN. As in the research of [9] NN generated the best prediction result comparing with the rest three. To further power up the prediction accuracy of the model, the same researcher conducted another research (Moro et al., 2014b). By combining historical transaction data, and conducting further feature engineering, Neural Network model yielded a significantly improved performance, AUC = 0.86 and ALIFT = 0.70.

NN, usually referred to as Artificial Neural Network (ANN), is a biologically inspired algorithm which is intended to simulate the way how our human brain works. NN is usually organized in layers. Each of the inputs is related to a neuron, and also the weight matrix is calculated. The layer includes weight matrix as well as neurons. The final output is the output layer. There are one or more hidden layers which transform the input into something that could be used by the output layer. [10]

The limitations of NN are, for one thing, it requires high computational power to train the network, for another thing is it is a "black box", in which researchers could only feed in data and study the output. There are some techniques can be used to fine-tune the output but cannot access to the core of decision-making process.

To solve this one study attempted to extract explanatory knowledge from the NN model by using a sensitivity analysis method, which can list out the rank of the inputs basing on the weight of the attribute, and test for the influence of the input on the data-driven model [9].

2.2.2 Extreme Gradient Boosting (XGBoost)

XGBoost is an optimized library of gradient boosting. According to the work of [11], it is a greedy machine learning technique for regression and classification. It could generate an optimized prediction model by rectifying a weak performing model. (Chen and Guestrin, 2016) has taken the theory one step further, they introduced a highly scalable end-to-end tree boosting algorithm and termed it as "XGBoost".

It is one of the most favourited algorithms used by Kaggle winner solutions [13] because it is fast in computation and usually perform with higher accuracy.

XGBoost could fully utilize the resource of a machine to achieve the capability of scaling. Besides processor and memory, it also utilizes disk space to process data. Two techniques are applied to achieve this: block compression and block sharding. The first technique compresses the rows and columns on the fly and the second technique distribute the data onto different disks and increase the efficiency of data processing and reading. (Chen and Guestrin, 2016)

XGBoost is an open-source package and is implemented available in python and R and some other popular programming language. Also, due to its scalability characteristic, XGBoost can compliant with Hadoop natively. Very recently, it extended compliance with JVM platforms such as Spark and Flink. (Chen and Guestrin, 2016)

In both of the previous attempts try to solve this direct marketing cold call challenge, XGBoost yielded the best performance comparing with all rest algorithms, with significantly high accuracy of 85% [14][15].

2.3 Data Overview

The dataset used in this research is collected by the Technical University of Munich, from real-world data from a bank in the United States. This bank intends to extend car insurance service to its existing customers. This bank organizes regular campaigns to extend its customer base. "Previous Attempt" attribute in the dataset indicating whether this customer has been contacted in the previous campaign and also the outcome is indicated in "outcome" attribute, with "1" indicating success and "0" indicating fail. This dataset has 4000 labelled records, and 18 input dimensions. Eleven of these variables are character type and seven of them are numeric type (see in Table 1). The output variable *CarInsurance* is binary (0 and 1) as for indicating whether the customer will eventually purchase the insurance. There are two major parts of information included in this dataset: One is customer's demographic information including - age, job, marital, education level, credit status, yearly balance so on so forth, another part is derived directly from the company's interaction record with the customer including communication method, last contact month, last contact day, call start and end time, days have passed since the previous contact and also the result when last time approach to this customer.

Table 1 Variable Type

No.	Column	Type
1	Id	categorical
2	Age	numeric
3	Job	categorical
4	Marital	categorical
5	Education	categorical
6	Default	categorical
7	Balance	numeric
8	HHInsurance	categorical
9	CarLoan	categorical
10	Communication	categorical
11	LastContactDay	categorical
12	LastContactMonth	categorical
13	NoOfContacts	numeric
14	DaysPassed	numeric
15	PrevAttempts	numeric
16	Outcome	categorical
17	CallStart	numeric
18	CallEnd	numeric
19	CarInsurance	categorical

3. Results and Discussion

3.1 Naïve Bayesian

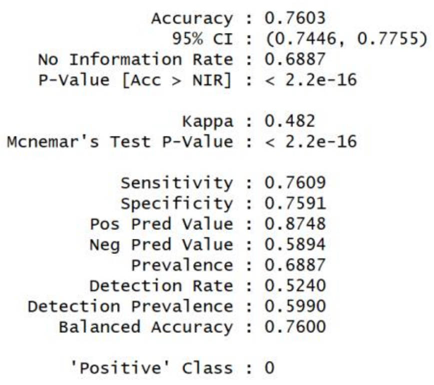
Naïve Bayesian model is built using "e1071" package. Firstly, a baseline model is built, then, "caret" package is combined for automatic model tuning, and lastly, a comparison model with one highly correlated variable dropped is built.

a. baseline model

Instead of the target variable, all rest of the variables are used to predict the target. The accuracy achieved is 0.7603 and AUC is 0.732 (see in Figure 1 and Figure 2). Also, kappa statistic value, which is denoted as kappa in confusion matrix output, is also referred to as an adjusted accuracy, taking into account the possibility of a correct prediction by chance along. The de facto standard for interpreting Kappa value is as follows:

- Poor prediction: < 0.2
- Fair prediction: between 0.2 and 0.4
- Moderate prediction: between 0.4 and 0.6
- Good prediction: between 0.6 and 0.8
- Very good prediction: 0.8 to 1

As indicated from the confusion matrix, Kappa value for the baseline model is 0.482, therefore, can conclude this is a moderate performing model. Since this is the basic model without any tuning, this result is expectable.



```
              Accuracy : 0.7603
                95% CI : (0.7446, 0.7755)
   No Information Rate : 0.6887
   P-Value [Acc > NIR] : < 2.2e-16

                 Kappa : 0.482
Mcnemar's Test P-Value : < 2.2e-16

           Sensitivity : 0.7609
           Specificity : 0.7591
        Pos Pred Value : 0.8748
        Neg Pred Value : 0.5894
            Prevalence : 0.6887
        Detection Rate : 0.5240
  Detection Prevalence : 0.5990
     Balanced Accuracy : 0.7600

      'Positive' Class : 0
```

Figure 1 confusion matrix output

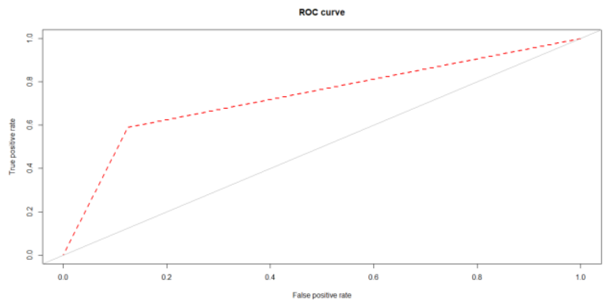
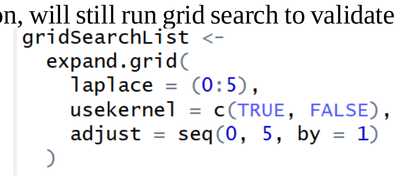


Figure 2 AUC

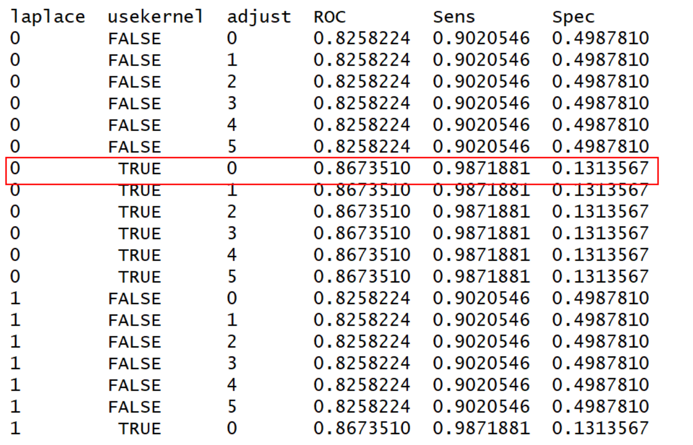
b. hyperparameter tuning with caret
"caret" package offers some parameter tuning. To look up what parameter can be managed by "caret", *modelLookup*() function can be used. For naïve_bayes model, three parameters are offered for tuning, laplace, usekernel, and adjust. As seen in Figure 3, will set the range for grid searching. Although as understand from data exploration, laplace is not required since there is no class showing zero probability, and kernel is required since all the numeric variables are not following Gaussian distribution, will still run grid search to validate the assumption.



```
gridSearchList <-
  expand.grid(
    laplace = (0:5),
    usekernel = c(TRUE, FALSE),
    adjust = seq(0, 5, by = 1)
  )
```

Figure 3 parameter setting

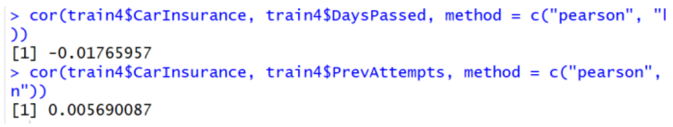
Part of the result is showing in Figure 4. As can tell from the final output, final optimized parameters are identified as *laplace* = 0, *usekernel* = TRUE and *adjust* = 0, which is in accordance with the assumption beforehand. Then, will apply the optimized parameter achieved to the initial model without any parameter tuning. An accuracy of 76.06% is obtained, no significant improvement found comparing with baseline model.



```
laplace  usekernel  adjust  ROC        Sens       Spec
0        FALSE      0       0.8258224  0.9020546  0.4987810
0        FALSE      1       0.8258224  0.9020546  0.4987810
0        FALSE      2       0.8258224  0.9020546  0.4987810
0        FALSE      3       0.8258224  0.9020546  0.4987810
0        FALSE      4       0.8258224  0.9020546  0.4987810
0        FALSE      5       0.8258224  0.9020546  0.4987810
0         TRUE      0       0.8673510  0.9871881  0.1313567
0         TRUE      1       0.8673510  0.9871881  0.1313567
0         TRUE      2       0.8673510  0.9871881  0.1313567
0         TRUE      3       0.8673510  0.9871881  0.1313567
0         TRUE      4       0.8673510  0.9871881  0.1313567
0         TRUE      5       0.8673510  0.9871881  0.1313567
1        FALSE      0       0.8258224  0.9020546  0.4987810
1        FALSE      1       0.8258224  0.9020546  0.4987810
1        FALSE      2       0.8258224  0.9020546  0.4987810
1        FALSE      3       0.8258224  0.9020546  0.4987810
1        FALSE      4       0.8258224  0.9020546  0.4987810
1        FALSE      5       0.8258224  0.9020546  0.4987810
1         TRUE      0       0.8673510  0.9871881  0.1313567
```

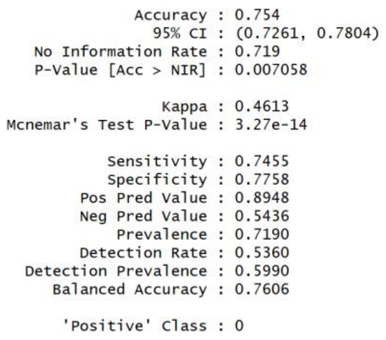
Figure 4 grid search for naïve Bayes

c. dropping off one high-correlated variable
As identified in the correlation matrix, two strongly correlated variables are: *DaysPassed* and *PrevAttempts*. To compare which of these two variables has a higher impact on the prediction model, will compute the correlation between each of them and the target variable. As seen in Figure 5, actually both of them has a weak impact on prediction, and *DaysPassed* showed even lower influence, therefore, decide to drop *DaysPassed.*
The result is shown in Figure 6, the accuracy of 0.7487 and 0.754 are achieved, which is slightly lower than the baseline model.



```
> cor(train4$CarInsurance, train4$DaysPassed, method = c("pearson", "l
))
[1] -0.01765957
> cor(train4$CarInsurance, train4$PrevAttempts, method = c("pearson",
n"))
[1] 0.005690087
```

Figure 5 correlation calculation



```
              Accuracy : 0.754
                95% CI : (0.7261, 0.7804)
   No Information Rate : 0.719
   P-Value [Acc > NIR] : 0.007058

                 Kappa : 0.4613
Mcnemar's Test P-Value : 3.27e-14

           Sensitivity : 0.7455
           Specificity : 0.7758
        Pos Pred Value : 0.8948
        Neg Pred Value : 0.5436
            Prevalence : 0.7190
        Detection Rate : 0.5360
  Detection Prevalence : 0.5990
     Balanced Accuracy : 0.7606

      'Positive' Class : 0
```

Figure 6 confusion matrix output

3.2 Decision Tree

In the first decision tree, "rpart" package is used. Like in Naïve Bayes, firstly, a baseline without any tuning will be built, then three extra experiments are attempted:

Table 2 Packages and the usage for building Decision Tree

No.	experiments	Package	Model
1	change parameter *minsplit, minbucket*	rpart	Decision Tree
2	use caret to tune cp		
3	change split method - entropy		

<u>Experiment 1</u>: *minsplit, minbucket* are tuned to prune the tree.
Minsplit **-** the minimum number of observations that must exist in a node
Minbucket **-** the minimum number of observations in any terminal <leaf> node.
Therefore, a small number choice of *minsplit, minbucket* will lead to complexed tree, whilst large number of *minsplit, minbucket* will get the tree pruned.
<u>Experiment 2:</u> use "caret" to tune cp.
cp is the complexity parameter; this is used to prune the complexity of the tree.
<u>Experiment 3</u>: change split method to "entropy"
Since the default split method is Gini, will change it to "entropy" as experiment on prediction accuracy.
a. baseline model
The baseline model is generated, and prediction method is chosen as "class". Use print(tree) function can print out the rules of the tree. As can tell from the result, the root has 3000 instances, with a proportion of 0.599 and 0.401 for class distribution. The first variable chosen is CallLast, which split the tree by 2037 versus 963. On the node CallLast < 0.00424, the purity is 0.7540, whilst on the node of CallLast > 0.00424 the purity is 0.7289. Next, *Outcome*, *LastContactMonth* and *HHInsurance* are selected in the following steps. As noticed, totally four variables are used in building the tree: *CallLast*, *Outcome*, *LastContactMonth* and *HHInsurance*. Figure 7 is the visualization of the decision tree.
The accuracy for this baseline tree is achieved as 0.8173for training data and 0.811 for testing data. As tell from the confusion matrix output in Figure 8, the Kappa value also increase to 0.6285 from Naïve Bayes. This indicates the decision tree built is a fairly good prediction.

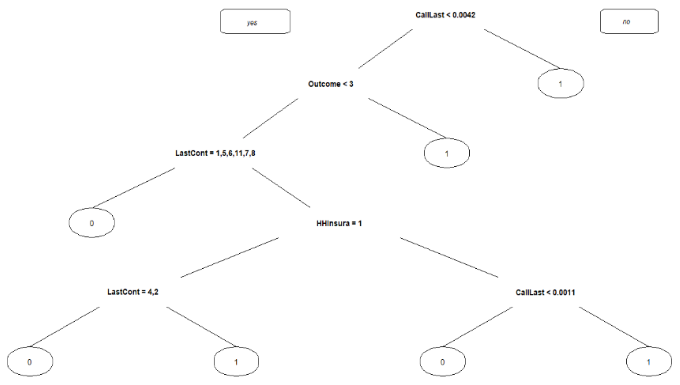
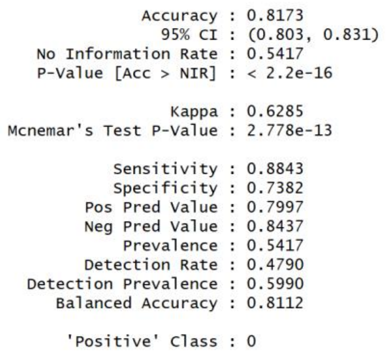


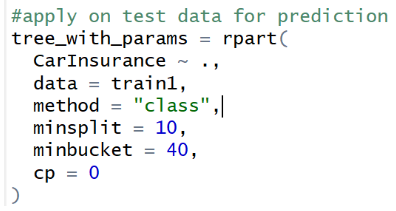
Figure 7 decision tree rules



```
               Accuracy : 0.8173
                 95% CI : (0.803, 0.831)
    No Information Rate : 0.5417
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6285
 Mcnemar's Test P-Value : 2.778e-13

            Sensitivity : 0.8843
            Specificity : 0.7382
         Pos Pred Value : 0.7997
         Neg Pred Value : 0.8437
             Prevalence : 0.5417
         Detection Rate : 0.4790
   Detection Prevalence : 0.5990
      Balanced Accuracy : 0.8112

       'Positive' Class : 0
```

Figure 8 confusion matrix for decision tree

b. change parameter *minsplit, minbucket*
Figure 9 is showing the code for tuning the parameter *minsplit, minbucket*. In this situation minimum split is defined as 10 and minimum bucket as 40, also complexity value is set as 0. Prediction result is showing 0.8347 of accuracy on training data and 0.818 accuracies on test data, which are both slightly better than baseline.



```
#apply on test data for prediction
tree_with_params = rpart(
  CarInsurance ~ .,
  data = train1,
  method = "class",
  minsplit = 10,
  minbucket = 40,
  cp = 0
)
```

Figure 9 change parameter minsplit, minbucket

c. use "caret" to tune cp
By using *plotcp()* function, can plot out the complexity of the existing decision tree. As can tell from Figure 10, when the complexity of the tree is roughly between 0.01 and 0.02, the tree is showing a relatively good performance, with low error rate and adequate tree size. Therefore, in grid search value is selected between -1 and 2, with a step of 0.01. The search result is shown with optimized cp identified as 0.

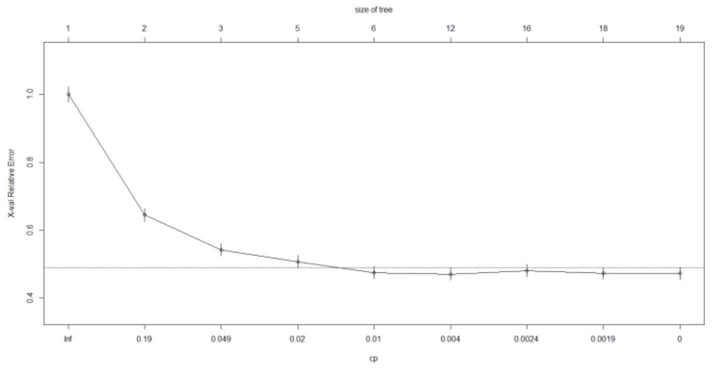
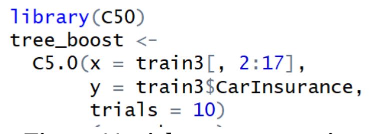


Figure 10 complexity plot

d. Split method change to entropy
Since the default split method is Gini, will try with Entropy split method and see whether there will be an improvement on performance. The accuracy is achieved as 0.6517 for training data and 0.796 for testing data.

3.3 Decision Tree with boosting

The second decision tree is built using "C50" package. Comparing with "rpart", "C50" offers the option to tune the parameter *trials*, which enables a boosting procedure. This method is very similar to other boosting tree methods such as AdaBoost.
As seen in Figure 11, the trial is specified as 10, this means 10 decision trees will be used in the boosted team for getting an optimized tree. According to the previous research result, the estimated improvement in prediction error rate is about 25%.
According to the confusion matrix for training and testing data, in this research, the boosting tree significantly increased training dataset prediction accuracy, but the effect on the testing set is not that significant. However, overall accuracy increased to 0.8796 from 0.8551.



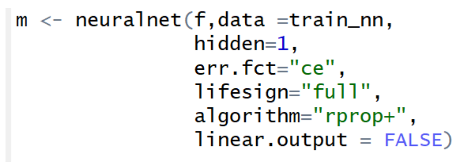
```
library(C50)
tree_boost <-
  C5.0(x = train3[, 2:17],
       y = train3$CarInsurance,
       trials = 10)
```

Figure 11 trial parameter tuning

3.4 Neural Networks

Neural networks algorithm can be used to do regression as well as classification. The strong point about neural networks is that it does not have any a priori assumption in the dataset. It is flexible to apply and can generate very good prediction result if tuned well. However, the drawback is the computational time it takes to train the neural is significantly higher than the previous models.

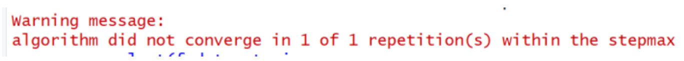
a. baseline model with one hidden layer and one neuro
Package "neuralnet" is used to train a basic neural network. As seen in below Figure 12, parameter *hidden* is specified as 1, indicating this is a neural network with one hidden l layer and one single neuro.



```
m <- neuralnet(f,data =train_nn,
               hidden=1,
               err.fct="ce",
               lifesign="full",
               algorithm="rprop+",
               linear.output = FALSE)
```

Figure 12 baseline for neural network

The result in Figure 13 is showing, the training process is not converging, and the max steps has exceeded. Therefore, to solve this problem has to increase the complexity of the model, this means either has to add-in extra hidden layer or hidden neurons.

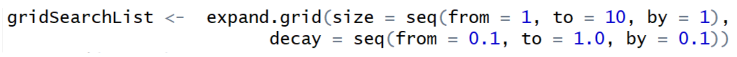


```
Warning message:
algorithm did not converge in 1 of 1 repetition(s) within the stepmax
```

Figure 13 result for single neuro neural network

b. A neural network with 3 neurons
The second model is built with 3 neurons in one hidden layer. This time, the training process managed to converge. With roughly 2 mins training time. Then, by applying the *plot()* function, the final network structure can be generated. As seen in Figure 14, as expected, there is one hidden layer between the input layer and the output layer. Each of the input variables is taken as one input neuron and the final prediction is generated with output neuro. Weights are calculated between each input neuro and each of the neuro in the hidden layer.
Then, this trained model is applied to training and testing data to compute the prediction. The output of the neural network is a value between 0 and 1. An activation function is defined as less than 0.5 then categorized as 0, and greater than 0.5, then classed as 1.
Then confusion matrix for training and testing data is calculated showing the accuracy of both training and testing data is very low, 0.677 and 0.569 respectively.

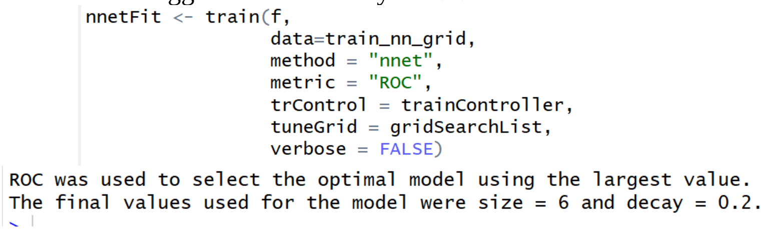
c. grid search with caret
Same as in decision tree and naïve Bayes, "caret" package is used to tune the parameters. The size is referring to the number of neurons in the hidden layer, and the decay parameter is referring to the weight. As seen in Figure 14, grid search is set with size and decay specified as below.



```
gridSearchList <-  expand.grid(size = seq(from = 1, to = 10, by = 1),
                               decay = seq(from = 0.1, to = 1.0, by = 0.1))
```

Figure 14 grid search for the neural network

Then the grid search is applied to train the model. The result and tuning code is shown in Figure 15. An optimized number of 6 neuro is suggested with a decay of 0.2.



```
nnetFit <- train(f,
                 data=train_nn_grid,
                 method = "nnet",
                 metric = "ROC",
                 trControl = trainController,
                 tuneGrid = gridSearchList,
                 verbose = FALSE)
```

```
ROC was used to select the optimal model using the largest value.
The final values used for the model were size = 6 and decay = 0.2.
```

Figure 15 tuning code and result

d. building the neural network with optimized tuning parameter
Next, an optimized neural network is built with parameters found in grid-search. The final accuracy for training and testing data is achieved as 0.6366 and 0.646, which is slightly better than the previous model but still performing very poor. Also, it is noticed that the training time consumed as raised to 5.12 mins.

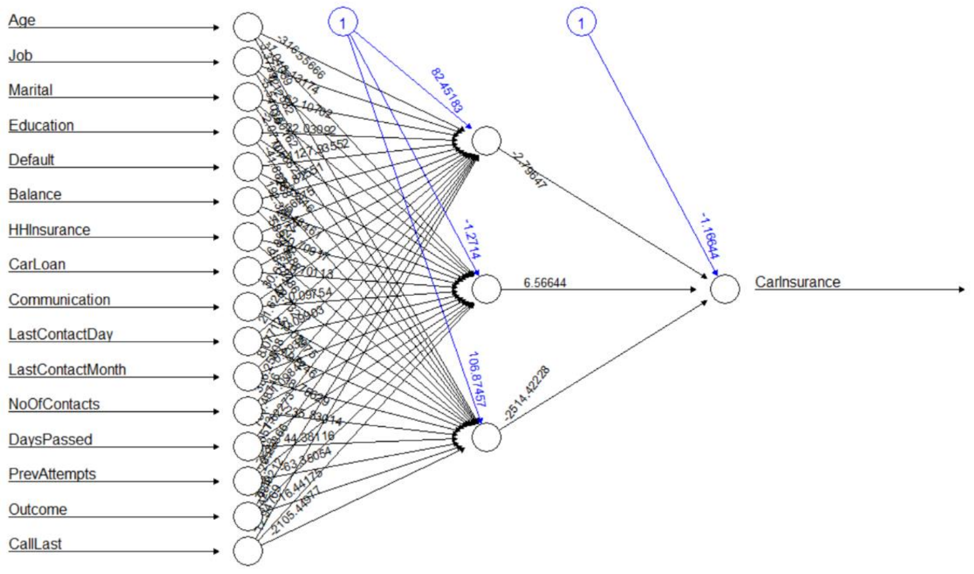


Figure 16 neural network structure

3.5 Analysis of the Evaluation

a. There are three algorithms picked, and four models are built in the previous experiment section. As shown in Table 3, the best performing model without any further effort on tuning is the decision tree model, which reaches an average accuracy of 0.8551. This result also matches with the findings in the previous literature review study. Many of the researches shown that decision tree can generate relatively good prediction in the classification task. One experiment did on the same dataset using the decision tree, also generated an accuracy of 0.82.
Naïve Bayesian is providing moderating prediction accuracy, not as good as decision tree, but generally acceptable. Whilst the ANN model has shown performance relatively low, which might due to the lack of training and low complexity of the model.

Table 3 Baseline model comparison for all models

		baseline		
No.	**Model**	**Train - Accuracy**	**Test - Accuracy**	**Average - Accuracy**
1	Naïve Bayes	0.7603	0.772	0.76615
2	decision tree1	0.8173	0.811	0.81415
3	decision tree2	0.8913	0.819	0.85515
4	ANN	0.6773	0.569	0.62315

b. The result after grid-search
For each of the model, "caret" package is used to do automatic tuning of the model. Except for the Naïve Bayesian model which remained almost the same as the baseline, all rest models are showing slight improvement. However, ANN is still giving poor performance. This might due to the learning rate set in grid-search, has not reached the optimized combination for ANN to give a good performance.

Table 4 Optimized models with tuned parameters

No.	Model	Train - Accuracy	Test - Accuracy	Average - Accuracy	optimized parameter
1	naiveBayes	0.7603	0.761	0.76065	laplace = 0, usekernel = TRUE and adjust = 0
2	decision tree1	0.8347	0.818	0.82635	cp = 0
3	decision tree2	0.8893	0.848	0.86865	trials = 30, model = rules and winnow = FALSE
4	ANN	0.6366	0.646	0.6413	size = 6 and decay = 0.2

c. additional experiments
There are some additional experiments did together with the baseline model as a comparison. In Decision Tree1, an experiment is done on changing the split index from Gini to Entropy as well as altering the parameter value for *minsplit, minbucket*. In both of the experiments, no significant improvements noticed. This might be caused by the *minsplit, minbucket* manually set by us is not the optimized ones. Further experiments required on tuning this.
Decision Tree2, with boosting is the best performing one, which is showing accuracy at 92.63% for training data and 83.3% for testing data. Although the average performance still very good, but the drop from above 90 to 83 also indicates the drawback of the decision tree, that it could generate high accuracy model base on training data but result in overfitting and cause higher error rate in testing data.
The last one is on dropping one highly correlated variable from the dataset. This might due to the reason there are only 19 input

variables when dropping one variable, more information is losing than the prediction strength gained by the algorithm.

Table 5 addition experiments on the model

No.	experiments	Package	Model	Train - Accuracy	Test - Accuracy	Average - Accuracy
1	change parameter minsplit , minbucket	rpart	Decision Tree1	-	-	-
2	change split method - entropy	rpart		0.6517	0.796	0.72385
3	boost tree	C50	Decision Tree2	0.9263	0.833	0.87965
4	drop high-correlated variable	e1071	Naïve Bayes	0.7487	0.754	0.75135

4. Conclusions

As the result shown in experiments and comparison analysis did, it is noticeable that for this dataset when applying classification prediction, the decision tree can provide more steady performance. As an attempt to optimize the decision tree model, the *trial* parameter is used to build a boosting tree. This largely increased the accuracy of prediction for training data to 92.63%, however, the testing dataset is not performing similarly well, only generating accuracy of 83.3%. This unveils the drawback of the decision tree that it is prone to overfitting. The advantage of the decision tree is that it is also easy to interpret and can be interpreted as rules. In the rule generated in this decision tree as shown in Figure 23, variables are selected according to their significance to the prediction model. Therefore, four important variables achieved. These four variables can be used, to build a more concise model instead of all the original input variable from the dataset. In this sense, the decision tree can be used for feature reduction.

ANN is showing low prediction rate, which might due to the complexity of the model. The first model we built is with 3 neurons, when increased that to 6 neurons, the performance slightly increased, but still not satisfying.

Further experiments are required on grid searching more optimized hyperparameter to fit the model well for ANN, and also pruning the decision tree complexity to solve the problem of overfitting.

References

[1] L. Tatiana, “Machine learning in Insurance,” *Accent. Rep.*, no. 1, pp. 1–36, 2015, doi: 10.1002/9781119183600.

[2] Bernard Marr, “How Big Data Is Changing Insurance Forever,” 2015. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2015/12/16/how-big-data-is-changing-the-insurance-industry-forever/#3295cae3289b. [Accessed: 09-Oct-2018].

[3] S. Tavish, “What Is The Difference Between Machine Learning &amp; Statistical Modeling,” 2015. [Online]. Available: https://www.analyticsvidhya.com/blog/2015/07/difference-machine-learning-statistical-modeling/. [Accessed: 14-Oct-2018].

[4] Asare-Frempong, J. and Jayabalan, M., 2017, September. Predicting customer response to bank direct telemarketing campaign. In 2017 International Conference on Engineering Technology and Technopreneurship (ICE2T) (pp. 1-4). IEEE.

[5] L. Rokach, *Data Mining with Decision Trees: Theory and Applications*, 2nd ed. World Scientific Publishing Co. Pte. Ltd., 2015.

[6] M. Karim and R. M. Rahman, “Decision Tree and Naïve Bayes Algorithm for Classification and Generation of Actionable Knowledge for Direct Marketing,” *J. Softw. Eng. Appl.*, vol. 06, no. 04, pp. 196–206, 2013, doi: 10.4236/jsea.2013.64025.

[7] Scikit-learn, “RBF SVM Parameters — scikit-learn 0.14 documentation,” *scikit-learn developers*, 2013. [Online]. Available: http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html. [Accessed: 14-Oct-2018].

[8] S. and Moro, P. and Cortez, and P. Rita, “A data-driven approach to predict the success of bank telemarketing,” *Decis. Support Syst.*, vol. 62, no. February 2014, pp. 22–31, 2014, doi: 10.1016/j.dss.2014.03.001.

[9] S. Moro, P. Cortez, and P. Rita, “Using customer lifetime value and neural networks to improve the prediction of bank deposit subscription in telemarketing campaigns,” *Neural Comput. Appl.*, vol. 26, no. 1, pp. 131–139, 2014, doi: 10.1007/s00521-014-1703-0.

[10] M. T. Hagan, H. B. Demuth, and M. H. Beale, “Neural Network Design,” *Bost. Massachusetts PWS*, vol. 2, p. 734, 1995, doi: 10.1007/1-84628-303-5.

[11] J. H. Friedman, “Greedy Function Approximation- A Gradient Boosting Machine,” *Statistics (Ber).*, vol. 29, no. 5, pp. 1189–1232, 2001, doi: doi:10.1214/aos/1013203451.

[12] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” 2016, doi: 10.1145/2939672.2939785.

[13] C. Tianqi, “Story and Lessons Behind the Evolution of XGBoost,” 2016. [Online]. Available: https://homes.cs.washington.edu/~tqchen/2016/03/10/story-and-lessons-behind-the-evolution-of-xgboost.html. [Accessed: 15-Oct-2018].

[14] B. Manikandan, “Cleaning, Visualizing and Modeling Cold Call Data | Kaggle,” *Kaggle*, 2017. [Online]. Available: https://www.kaggle.com/manibhask/cleaning-visualizing-and-modeling-cold-call-data. [Accessed: 08-Oct-2018].

[15] Emma Ren, “Cold Calls: Data Mining and Model Selection | Kaggle,” *Kaggle*, 2017. [Online]. Available: https://www.kaggle.com/emmaren/cold-calls-data-mining-and-model-selection. [Accessed: 08-Oct-2018].