Predicting If A Driver Will File An Insurance Claim

Biswadeep Kumar Ghoshal

(Under the Supervision Of Prof. Atanu Kumar Ghosh)

Contents

Chapter 1. Introduction 4
1.1. Objective 4
1.2. Data Description 4

Chapter 2. Data Modification and Pre-Processing 5

Chapter 3. Methodology: Model Fitting 6
3.1. Parametric Model: Logistic Regression 6
3.2. Non-Parametric Models 11

Chapter 4. Discussion 18

Acknowledgement 19

Appendix 20

Bibliography 23

CHAPTER 1

Introduction

Car drivers usually file insurance claims for their cars to safeguard against the cost of repairing in case of damage to the car. The damage is done mostly when the drivers resort to rash driving. So, it is ideal that drivers who drive safely file lesser insurance claims than those who do not drive safely.

However, it is often seen the other way round. Drivers who drive safely are often found to file insurances at higher costs. Inaccuracies in car insurance company's claim predictions raise the cost of insurance for good drivers and reduce the price for bad ones. Hence, good drivers sometimes end up filing higher insurance claims than the bad ones.

This is one of the issues which Porto Seguro, one of Brazil's largest auto and homeowner insurance companies, wants to address. It wants to improve upon its claim predictions and provide insurance to car drivers as suited to their needs.

1.1. Objective

Our task is to help the drivers improve over this situation and resolve their issues regarding such inaccuracies in claim predictions made by the insurance companies. Here, we are presented with data on some of such auto insurance policy holders. Our objective is to fit a model based on this data and use that model to predict the probability that a driver will file an auto insurance claim.

1.2. Data Description

The entire data set for our study is provided in two parts. One is the training data set, and the other is the test data set. The training data set consists of $5,95,212$ observations on 59 variables, and the test data set comprises $8,92,816$ observations on 58 variables. We are to fit a model based on the training data set and use it to predict the values of the response in the test data.

In both the training and test data sets, each row or observation corresponds to an auto insurance policy holder. Variables that belong to similar groupings are tagged as such in the variable names (e.g., "ind", "reg", "car", "calc"). In addition, variable names include the post-fix "bin" to indicate binary variables and "cat" to indicate categorical features. Variables without these designations are either continuous or ordinal. In each data set, some observations have values of -1 for some variables, indicating that the values of those variables are missing for those observations. The variable **target** is a binary variable indicating whether an auto insurance holder has filed an insurance claim. If a driver files an insurance claim, the value of **target** for the driver is 1, otherwise, it is 0.

Here, target is our response variable and we wish to fit a model to predict $P(target = 1)$ for an auto insurance holder.

CHAPTER 2

Data Modification and Pre-Processing

Here, we note that the test dataset we have does not contain values of the response for us to evaluate the performance of any model we may fit. So, instead, we divide our training dataset randomly into two parts, one part for fitting the model and the other part for evaluating the performance of the model in terms of the test error. Since our response **target** is binary, we can fit a logistic regression model to this data. We may also fit a suitable non-parametric model. Before proceeding to fit any such model, we note that, as stated before, the data set provided for our study contains some observations whose values are missing for some variables. So, we first remove these observations from both the training and the test data sets.

The original data set contained a variable id which corresponds to the identity of an individual driver. Since it is not necessary for fitting our model, we have not included it in the modified data sets. It is the only variable which has been discarded from both the datasets. Now, the matrix of covariate values in both the data sets has each column representing a covariate. Some of these covariates are categorical. They should be treated in R accordingly. R treats such categorical covariates as factors. So, finally, we modify the covariate matrices to treat our categorical covariate values as factors.

Thus, we have suitably set or response and the matrix of covariates. We can now fit a model based on this data set.

CHAPTER 3

Methodology: Model Fitting

3.1. Parametric Model: Logistic Regression

We may, first, start by fitting a parametric model to the data. Our response **target** is a binary variable. Thus we can fit a logistic regression model of the form

$$\log(\frac{p_x}{1-p_x}) = \beta_0 + \beta^T x$$

where $p_x = P(target = 1|x)$, x being a vector of covariate values for an observation

β_0 =Intercept part of the model, and

β =Vector of coefficients corresponding to the covariates

Now, our training dataset contains data for 74958 car drivers on 57 covariates. This number of covariates, however, is quite large. Moreover, if we fit our model based on all the 57 covariates, we are faced with a variety of problems with having a large number of covariates:

- **Multicollinearity:** One of the foremost problems with having a large number of covariates in our model is the presence of multicollinearity. If we have a large number of covariates, then, more than often, it is found that some of the covariates are dependent on others. Hence, the estimated coefficients of those covariates are unidentifiable and the standard errors of the estimates become very high.
- **Lower Prediction Accuracy:** Since we include a large number of covariates in our model, the model will consist of a large number of parameters. Although this model may exhibit low bias, the variance of the estimated coefficients will be higher, and this will lead to lower prediction accuracy for the model.
- **Hard Interpretation:** It often becomes hard to interpret a model with a large number of covariates. Most covariates are found to have significantly less effect on the response in this case.

So, it is necessary for us to fit a model based on a reduced number of covariates. There are many ways this can be achieved. We can apply principal component technique to the covariates and choose $k(\ll 57)$ linear combinations of covariates and fit the model based on these linear combinations. But sometimes, it is hard to interpret the model. Also. it is difficult to understand what a linear combination of the covariates signifies. Again, we can reduce dimension of the data also by choosing k covariates with the largest correlation with the response. We may also adopt another technique which selects a subset of the 57 covariates. So, we can reduce the number of covariates by the following ways:

- **Variable Selection:** This involves selecting a subset of the 57 covariates which are related with the response and fitting our model based on those covariates
- **Principal Component:** This technique involves computing $k(\ll 57)$ different linear combinations, or projections, of the covariates, which we can then use as covariates to fit the model.
- **Method of Shrinkage:** This technique fits a regression model with all the covariates, but the estimates of the covariate coefficients are shrunken towards 0 in a suitable way. Depending on the type of shrinkage method, the coefficient estimates for some covariates may become exactly 0 and hence the corresponding covariates can be eventually dropped off the model. Thus shrinkage can also perform variable selection.

Here, we choose to adopt the method of shrinkage for selecting the covariates for our model fitting. There are two methods of shrinkage, viz, Ridge Regression and LASSO. We perform LASSO as it is the only shrinkage method that performs variable selection.

3.1.1. LASSO. Normally, fitting a generalized linear model with y as response and X as the set of all the covariates requires estimating the model parameters β_0 and $\beta = (\beta_1, ..., \beta_p)^T$ by maximing the likelihood function, or equivalently, the log-likelihood function, of the model parameters given the data, with respect to the parameters, i.e, we maximize the log-likelihood function

$$\mathcal{L}(\beta_0, \beta; y, X)$$

with respect to β_0 and β, where the functional form of $\mathcal{L}$ depends on the nature of response y and the type of model to be fitted.

This is same as minimizing the negative log-likelihood function

$$-\mathcal{L}(\beta_0, \beta; y, X)$$

or, equivalently

$$-\frac{1}{n}\mathcal{L}(\beta_0, \beta; y, X)$$

where n =Number of observations in the data set.

However, in LASSO, the model parameters β_0 and β are estimated by minimizing

$$\phi(\beta_0, \beta) = -\frac{1}{n}\mathcal{L}(\beta_0, \beta; y, X) + \lambda \parallel \beta \parallel_1$$

. where

$$\parallel \beta \parallel_1 = \sum_{j=1}^{p} |\beta_j|$$

and λ is a non-negative quantity which controls the amount of shrinkage to be performed. It is called a shrinkage parameter for LASSO. Greater the value of λ, the more the shrinkage is performed. If we choose $\lambda = \infty$, then, $\beta_j = 0$ for every $j = 1(1)p$. Again, if $\lambda = 0$, we essentially have the problem of minimizing the negative log-likelihood only.

Thus, LASSO performs restricted minimization of the negative log-likelihood to obtain the estimates $\hat{\beta}_0$ and $\hat{\beta}$. The estimates are obtained so that

$$\hat{\beta}_j = 0 \qquad \text{for some covariates } x_j$$

. Here, our response $y = target$ is binary, and we wish to fit a logistic regression model. So, our objective function becomes

$$\phi(\beta_0, \beta) = -\frac{1}{n}\sum_{i=1}^{n}\{y_i log_e(P(y=1|x_i)) + (1-y_i) log_e(P(y=0|x_i))\} + \lambda \parallel \beta \parallel$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\{y_i(\beta_0 + \beta^T x_i) - log_e(1 + e^{\beta_0 + \beta^T x_I})\} + \lambda \parallel \beta \parallel$$

The objective function is convex but not differentiable with respect to β. Hence, there is apparently no exact solution to this restricted minimization problem. However, this minimization problem can be solve iteratively.

There are many such iterative solutions to this minimization problem. One such solution is obtained by running **cyclical coordinate descent** method, which successively minimizes the objective function with respect to one $\beta_j, j = 1(1)p$, keeping the other β_j's fixed, and then cycles repeatedly through other parameters until it reaches convergence.

Thus, our fitted model is

$$\log(\frac{\hat{p}_x}{1-\hat{p}_x}) = \hat{\beta}_0 + \hat{\beta}^T x$$

where, $\hat{p_x}$=estimate of $P(target = 1|x)$ for an observation with covariate values x,

and $\hat{\beta} = (\hat{\beta}_1, ..., \hat{\beta}_p)^T$ is such an estimate of $\beta = (\beta_1, ..., \beta_p)^T$ that $\hat{\beta}_j = 0$ for some $j = 1(1)p$.

Here, we note that the fitted model we obtain above corresponds to a specific value of the shrinkage parameter λ. For different values of λ, we get different fitted models for the same dataset.

An "optimum" choice of λ is obtained from the model which accounts for the minimum cross-validation error or test set error, if available.

Another optimum choice of λ can be made from the model whose amount of cross-validation error or test set error is the greatest within upper one standard error of the minimum cross-validation error or test set error. For e.g, let us suppose $CV(\lambda)$ is the cross-validation error of a fitted model with shrinkage parameter λ. Then, we minimize $CV(\lambda)$ wrt λ to get

$$\hat{\lambda} = \arg\min_{\lambda} CV(\lambda)$$

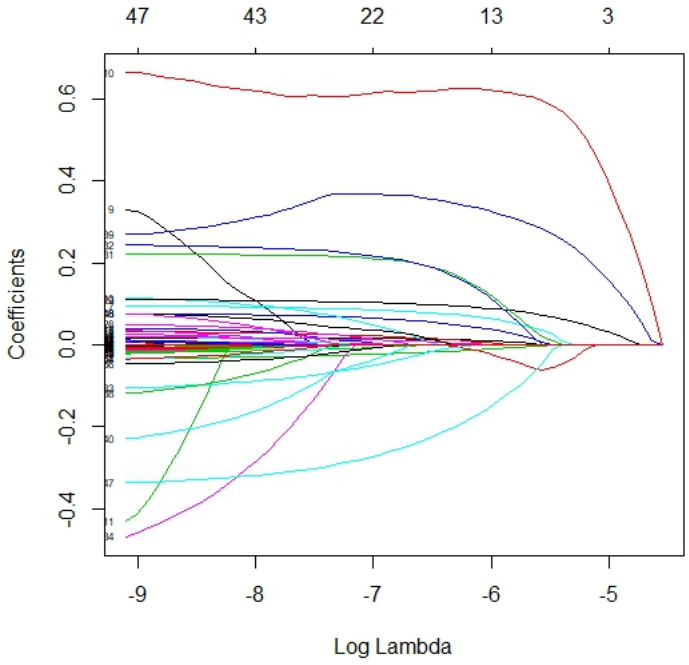
Then, we choose the fitted model with corresponding shrinkage parameter λ_{SE} such that its cross-validation error

$$CV(\lambda_{SE}) \leq CV(\hat{\lambda}) + SE(CV(\hat{\lambda}))$$

We sometimes adopt the second procedure to make the fitted model more simple. This approach is also known as the 1 Standard Error Rule. In this way, we use LASSO to fit a parametric model to the data.

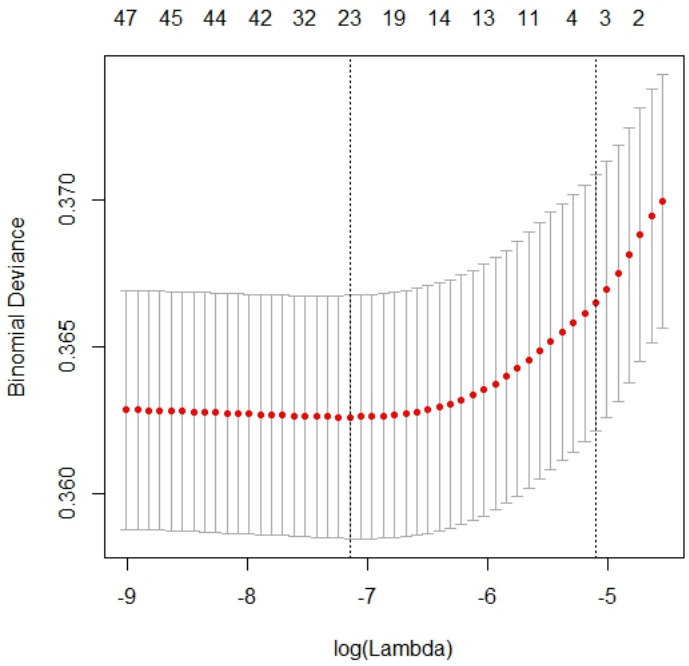
We can, similarly, adopt the above idea of LASSO to fit a logistic regression model to our data set such that the fitted model contains some of the 57 covariates. We first fit several models for different values of λ and plot their coefficient paths

FIGURE 3.1.1. Coefficient Paths



Now, to find the optimum model, we now plot the cross-validation errors for the different models we obtain corresponding to the different values of λ:

FIGURE 3.1.2. Cross Validation Curve



We now obtain the value of λusing the 1 Standard Error Rule and thereby fit our desired parametric model. The corresponding model has been found to contain only 3 predictors, viz, "ps_car_13", "ps_ind_05_cat" and "ps_ind_17_bin". So, LASSO has significantly reduced the number of covariates. The model has a test error of 0.04524443 and the predicted probabilities for individuals filing insurance are given below along with the confusion matrix:

Predicted Probabilities: $0.04380148, 0.0557762, 0.0454092, 0.05099336, 0.04229342, ...$

TABLE 1. Confusion Matrix for LASSO

		Original	
		0	1
Predicted	0	47712	2261
	1	0	0

It should be noted that, although we used LASSO to fit a model with some of the covariates, LASSO cannot always remove multicollinearity while fitting a model. However, we know that RIDGE Regression can remove multicollinearity. Therefore, on one hand, LASSO performs variable selection only and on the other hand, RIDGE can only remove multicollinearity. So, we want a compromise between the two, with more emphasis on LASSO for variable selection. This can be

achieved by Elasticnet Regularization. This minimizes

$$\phi(\beta_0, \beta) = -\frac{1}{n}\mathcal{L}(\beta_0, \beta; y, X) + \lambda[(1-\alpha)\frac{\| \beta \|_2^2}{2} + \alpha \| \beta \|_1]$$

where $\alpha \epsilon [0, 1]$.

It can be immediately seen that $\alpha = 0$ and $\alpha = 1$ respectively correspond to RIDGE and LASSO. Since we want this minimization problem to behave more like LASSO, we choose $\alpha = 1-\epsilon$ where ϵ is a very small positive quantity. So, similar to what we did using LASSO, we fit several models via the Elasticnet Regularization for several values of λ and choose the optimum model among them. Doing so, gives us a model with a test error of 0.04524443 and predicted probabilities along with the confusion matrix given below:

Predicted Probabilites: $0.02715734, 0.028655, 0.02814708, 0.02764791, 0.02565982, ...$

TABLE 2. Confusion Matrix for Elasticnet Regularization

		Original	
		0	1
Predicted	0	47712	2261
	1	0	0

Now, we fitted the above model assuming the underlying model to be of the form.

$$\log(\frac{p_x}{1-p_x}) = \beta_0 + \beta^T x$$

However, this above assumption may not always hold true. The model we fitted above was obtained via the parametric approach, i.e, assuming a parametric form for the underlying model given above. Also, this model does not predict any probability to be greater than 0.2. Hence, this model is not so reliable in terms of prediction. So, we may wish to now fit a model without assuming any parametric form. Therefore, we now proceed to fit a suitable non-parametric model to the given data.

3.2. Non-Parametric Models

3.2.1. Fisher's Linear Discriminant Function. Suppose we have the problem of predicting a class label Y based on p covariates $X_1, X_2, ..., X_p$, where Y assumes two values, viz, 0 or 1. Thus, we need to classify an observation based on its covariate values $X_1, X_2, ..., X_p$ to one of the two classes, say, C_1 and C_2. We want to discriminate between the two classes using a linear function $l^T X$ of the covariate values. Suppose $l^T X$ has some distribution $f(l^T\mu_1, l^T\Sigma l)$ under the class C_1 and another distribution $f(l^T\mu_2, l^T\Sigma l)$ under the class C_2. The more widely different the two class distributions are, the greater is the discriminating power of $l^T X$. We may take $\frac{(l^T(\mu_1-\mu_2))^2}{l^T\Sigma l}$ as a measure of difference between the two class distributions.

It can be shown that $\frac{(l^T(\mu_1-\mu_2))^2}{l^T\Sigma l}$ has the maximum value when $l \propto \Sigma^{-1}(\mu_1 - \mu_2)$. The linear discriminating function is therefore,

$$l^T X = (\mu_1 - \mu_2)^T \Sigma^{-1} X$$

We classify an observation with covariate value X' to C_1 if $(\mu_1 - \mu_2)^T \Sigma^{-1} X'$is closer to $(\mu_1 - \mu_2)^T \Sigma^{-1} \mu_1$ than to $(\mu_1 - \mu_2)^T \Sigma^{-1} \mu_2$. Since we do not know the values of the parameters μ_1, μ_2, Σ, we instead work with their respective plug-in-estimates $\bar{X}_1, \bar{X}_2, S$ where

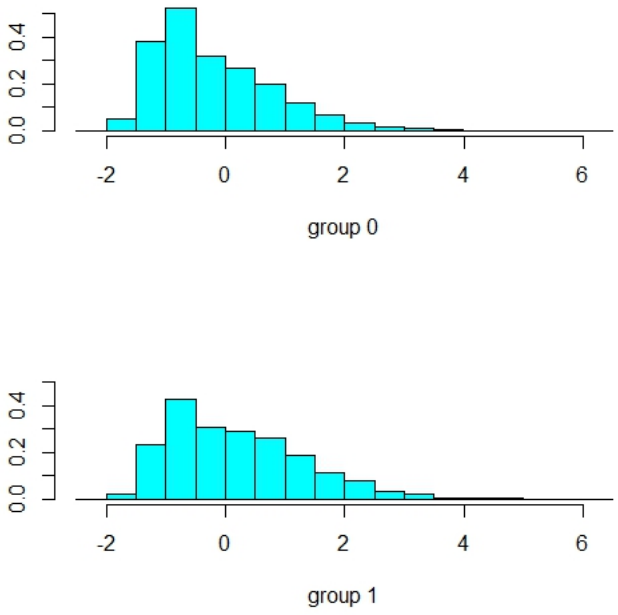
$\bar{X}_1$=Mean of covariate values in class C_1

$\bar{X}_2$=Mean of covariate values in class C_2

S=Pooled sample dispersion matrix

Here, our response *target* can be thought of as a class label, having values in $\{0, 1\}$. Thus, we can use linear discriminating function to build a classification model for our data.

FIGURE 3.2.1. LDA Plot



Applying LDA over our dataset, we get a model with test error of 0.04524443 with corresponding class probabilities and the confusion matrix given below:

Predicted Probabilities: $0.03827449, 0.06868686, 0.04179001, 0.0611869, 0.03513571$

TABLE 3. Confusion Matrix for LDF

		Original	
		0	1
Predicted	0	47712	2261
	1	0	0

3.2.2. Prediction Trees. Suppose we want to predict a response Y with the help of p predictors $X_1, X_2, ..., X_p$. We make predictions by growing binary trees. At first, at the root node, which contains all the observations, we ask a binary question (answer would be either a 'yes' or a 'no') about one of the p covariates $X_1, X_2, ..., X_p$. Depending on the answer to that question, two daughter nodes are created and at each node, we again ask a binary question about one of the covariates. This goes on until we reach a leaf node or terminal node of the tree, where we eventually make predictions about the response. This technique is useful when the dataset contains a large number of covariates which interact in non-linear complicated ways. This technique firstly partitions the sample space into smaller orthogonal regions where the interactions are manageable. These smaller regions are partitioned again, recursively, until we get regions $R_1, R_2, ..., R_m$ of the sample space which are so small that separate simple models can be fit to those regions. Our fitted model here is given by

$$\hat{f}_{tree}(x) = \sum_{j=1}^{m} c_j I(x \, \epsilon \, R_j)$$

for any covariate value x.

For a binary classification problem, $Y \, \epsilon \, \{0, 1\}$. Hence, for any $j = 1(1)m$, c_j can take any of the values 0 or 1. Much of our problem lies in choosing the covariates and the types of question to be asked at every stage, and also when to stop partitioning the dataset. The covariates $X_1, X_2, ..., X_p$ can be either continuous, categorical, discrete but ordered, or a combination of any number of the three types. Let us make our procedure of creating trees more formal.

While creating prediction trees, we have two problems to solve:

(1) Fixing the partitioned regions $R_1, R_2, ..., R_m$ of the dataset
(2) Making predictions at each of the partitions

Each of the problems has some optimality criteria to follow, which is maximizing the "information" regarding the response. Let us first decide how to create the partition of the dataset.

Partitioning the Dataset: Here, our response Y is categorical. So, we can use Information Theory to measure how much information we get to know about Y by knowing the value of a dicrete variable A, where A is an answer to the question regarding one of the p covariates $X_1, X_2, ..., X_p$. ,i.e, $A = I_{\mathcal{A}}(X)$, for a question $\mathcal{A}$. We define entropy of Y as.

$$H(Y) = \sum_{y} -P(Y = y) \log_2 P(Y = y)$$

We also define the conditional entropy of Y from knowing the value of A as

$$H(Y|A = a) = \sum_{y} -P(Y = y|A = a) \log_2 P(Y = y|A = a)$$

The first measure denotes the uncertainty involved in Y, whereas the second one denotes the uncertainty in Y after knowing that $A = a$.

Therefore, the measure

$$I(Y, A = a) = H(Y) - H(Y|A = a)$$

denotes the reduction in uncertainty about Y after knowing that $A = a$.

Thus,

$$I(Y, A) = \sum_a I(Y, A = a)P(A = a)$$

denotes how much the uncertainty of Y decreases, on an average, on knowing the value of A.

We choose that question A at our root node which amounts to maximum average reduction in uncertainty about Y, i.e, which maximizes $I(Y, A)$.

When we want to choose questions at subsequent nodes, we need to take into account what we already know. Mathematically,what this means is that if we reach the node when $A = a$ and $B = b$, we look for the question C which maximizes $I[Y, C|A = a, B = b]$.

In this way, we can create the partition $R_1, R_2, ..., R_m$ of the dataset.

Making Predictions: Since our response Y is binary, a classification tree can make two kinds of predictions, viz, class prediction and probability prediction. For probability predictions, each terminal node in the tree gives us a probability distribution over the classes. If the terminal node corresponds to the sequence of answers $A = a, B = b, ..., Q = q$, then ideally this would give us $P(Y = y|A = a, B = b, ..., Q = q)$ for each possible value y of the response, which is estimated by the empirical relative frequencies of the classes in that node. For class predictions, there are many methods for doing so, depending on the choice of the loss functions used. Since we are mainly interested in making probability predictions, we do not go into details on making class predictions.

Sometimes, we end up creating large trees for our predictions, or, especially, a full tree with each leaf node having only one observation. Such large trees often over fit the training data and are, therefore, not suitable for prediction purposes. So, instead of using a large tree or a fully grown tree, we often resort to using a sub-tree of the tree for prediction, which is obtained by pruning the tree from below. We choose that sub-tree that gives us the minimum cross-validation error or the test set error, if the test set is available, which is determined in terms of misclassification rate. Let T be a tree classifier having terminal nodes $\tau_1, \tau_2, ..., \tau_m$. Then the misclassification rate for the tree is given by

$$R(T) = \sum_{i=1}^{m} R(\tau_i)P(\tau_i)$$

where $R(\tau_i)$ is the misclassification rate for the node τ_i and $P(\tau_i)$ is the probability of an observation belong to node τ_i.

An estimate of the misclassification rate $R(T)$ is given by

$$\widehat{R(T)} = \sum_{i=1}^{m} r(\tau_i)p(\tau_i) = \sum_{i=1}^{m} \widehat{R(\tau_i)}$$

where $p(\tau_i)$ is the proportion of observations in the node τ_i.

It can be seen that $\widehat{R(T)}$ assumes smaller values for larger trees and hence is an optimistic estimate of the true misclassification rate $R(T)$. So, we start pruning the tree using the cost complexity pruning.

Let $\alpha(\geq 0)$ be a complexity parameter. Then, for a node τ in the tree T, we define a cost at the node τ as

$$C_\alpha(\tau) = \widehat{R(\tau)} + \alpha$$

Then, we define the cost complexity criterion for the tree T as

$$C_\alpha(T) = \widehat{R(T)} + \alpha|T|$$

where $|T|$ is the number of leaf nodes of T.

For each α, we choose a sub tree T_α of T_0, the initially fully grown tree, which minimizes the cost complexity criterion $C_\alpha(T)$, i.e, we choose T_α such that

$$T_\alpha = \arg\min_T C_\alpha(T)$$

Such a sub tree T_α is called a minimizing sub tree at α.

Suppose for $\alpha = \alpha_1$, we obtain a minimizing sub tree $T_1 = T_{\alpha_1}$ by pruning the tree T_0. As we keep increasing α, T_1continues being a minimizing sub tree until we get $\alpha = \alpha_2$, where we obtain another minimizing sub tree $T_2 = T_{\alpha_2}$, by pruning T_1. This goes on and finally we obtain a sequence of nested minimizing sub trees

$$T_1 \succ T_2 \succ T_3 \succ ... \succ T_m$$

corresponding to the complexity parameters

$$\alpha_1 < \alpha_2 < \alpha_3 < ... < \alpha_m$$

Now, we are to choose one minimizing sub tree among the sub trees above which minimizes the test set error or cross validation-error which is evaluated in terms of the misclassification error.

Minimizing Test Error: Suppose we have a test set at our hand, which consists of n' observations, say. We grow the tree T_0 from the learning set and prune it to get the sequence of nested sub trees $T_1, ..., T_m$. Then, we assign a class to each terminal node of every sub tree. We now apply the tree model over the n test observations and predict their classes. Now, since we know the true classes for the test observations, we can estimate the true misclassification rate $R(T_k)$ for a sub tree T_k by the $\widehat{R(T_k)}$, which is the proportion of test observations misclassified by the sub tree T_k, $k = 1(1)m$. We choose that sub tree which has the minimum misclassification rate, we choose a tree T_*such that

$$T_* = \arg\min_k \widehat{R(T_k)}$$

. The estimate of the standard error of estimated misclassification rate for this tree T_* is given by

$$\widehat{SE(\widehat{R(T_*)})} = \left\{ \frac{\widehat{R(T_*)}(1 - \widehat{R(T_*)})}{n'} \right\}^{\frac{1}{2}}$$

Minimizing Cross-Validation Error: In absence of a test set, we choose a sub tree of T_0which has the minimum cross-validation error. Suppose we do a $K-$fold cross-validation. Let us denote the training set and the test set in the k^{th} fold by $\mathcal{L}_k$ and τ_k respectively, $k = 1(1)K$. In the k^{th} fold, we fit a tree $T_0^{(k)}$ over the k^{th} learning set $\mathcal{L}_k$ and obtain the best pruned sub tree $T_\alpha^{(k)}$ of $T_0^{(k)}$ for a specific complexity parameter α. Then , we apply the model over the k^{th} test set τ_k and predict the class for each of the observations. Let, for every class j, $n_{ij}^{(k)}(\alpha)$ be the number of observations from the j^{th} class which the tree $T_\alpha^{(k)}$ classifies into the

i^{th} class. Then, the total number of observations from j^{th} class which have been classified into the i^{th} class in the entire K-fold cross- validation is

$$n_{ij}(\alpha) = \sum_{k=1}^{K} n_{ij}^{(k)}(\alpha)$$

Finally, for a given α, the cross-validated misclassification rate is given by

$$\widehat{R_{CV}(T_\alpha)} = \frac{1}{n} \sum_i \sum_j n_{ij}(\alpha)$$

. We choose that sub tree T_* such that

$$T_* = \arg\min_\alpha \widehat{R_{CV}(T_\alpha)}$$

However, we may wish to choose a more simple tree. This is given by the sub tree T_{**} of T_0 such that

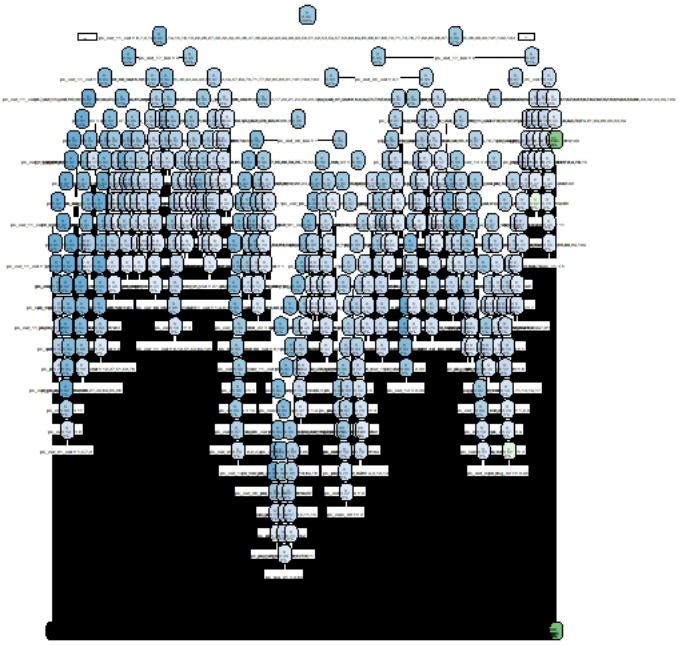
$$\widehat{R_{CV}(T_{**})} \leq \widehat{R_{CV}(T_*)} + \widehat{SE(R_{CV}(T_*))}$$

In this way, using either test set error or cross-validation error, we choose a tree which suitably fits the data.

Thus, for our given data, where the response is binary, we can fit a binary classification tree by the above procedure.

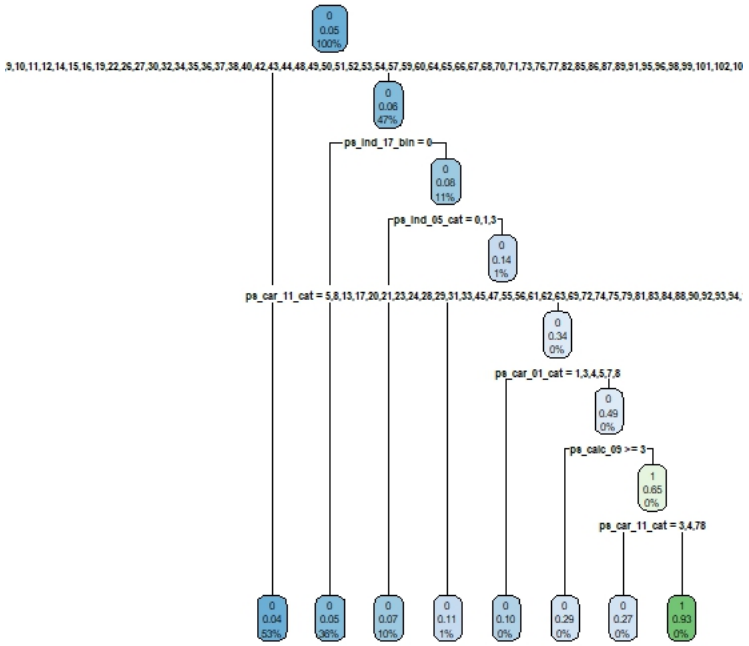
We start first by growing the full tree for our data

FIGURE 3.2.2. Full Classification Tree



We now prune back this tree using the cost complexity criterion to get our optimal tree given below:

FIGURE 3.2.3. Pruned Tree Obtained By Cost Complexity Criterion



Thus, we have obtained an optimum tree model with a test error of 0.04528445 and the predicted probabilities and the confusion matrix given below:

Predicted Probabilities: $0.03573395, 0.05032287, 0.05032287, 0.05032287, 0.05032287, ...$

TABLE 4. Confusion Matrix for Classification Tree

		Original	
		0	1
Predicted	0	47707	2258
	1	5	3

CHAPTER 4

Discussion

The models we have obtained in the previous chapter seem to perform fairly on our data in terms of test set error. However, from the confusion matrices for the respective models, it can be noted that except the classification trees, all other models predict the class labels for the test observation to be only 0. This is because almost all of the observations have a value of 0 for the response *target*. Only a handful of observations have a value of 1. None of the observations, whose true class is 1, is classified to that class by the other models. The classification tree overcomes this problem somewhat by classifying some observations, originally from the class 1, to that class. Moreover, the classification tree gives an almost same test error as the other fitted models. Thus, considering all these, we can conclude the classification tree to be an appropriate model for our given data.

Acknowledgement

I would like to thank my supervisor Prof. Atanu Kumar Ghosh for his support in this project. I would also like to thank our Head, Prof. Biswajit Roy, and the rest of the faculty members of the Department of Statistics, Presidency University, and also my friends for their support when needed.

Appendix

The following is the code in R which we have run to fit our desired models:

```
x=read.csv("porto-seguro-safe-driver-prediction\\train\\train.csv",header=T)            #---
z=read.csv("porto-seguro-safe-driver-prediction\\test\\test.csv",header=T)              #---
x[1:30,1:6]                    #--- Viewing the abridged training set
```

```
##---Removing observations with missing values from the datasets---##
missing=function(vec)          #--- To determine whether vec has a value -1
{
        return(-1 %in% vec)
}

miss_train=apply(x,1,missing)
miss_test=apply(z,1,missing)
omit=which(miss_train==T)
omit2=which(miss_test==T)
x_mod=x[-omit,]          #--- training set after discarding observations with missing values
z_mod=z[-omit2,]               #--- test set after discarding observations with missing valu
```

```
set.seed(1)
N=nrow(x_mod)
train=sample(N,floor(0.6*N))
train_set=x_mod[train,-1]              #---excluding the covariate "id" as it corresponds to
test_set=x_mod[-train,-1]              #---excluding the covariate "id" as it corresponds to


y=train_set[,1]                #---training response
X=train_set[,-1]                       #---training covariates
Z=test_set                             #---test covariates
```

```
library(methods)
iden=function(mat,sep)         #--- to identify which columns of 'mat' are categorical
{
        vec=c()
        for(j in 1:ncol(mat))
        {
                char=unlist(strsplit(colnames(mat)[j],split=sep))
                if("cat" %in% char||"bin" %in% char)
                {
```

```
                        vec=c(vec,j)
                }
        }
        return(vec)
}
v=iden(X, sep="_")      #--- identifying the training categorical covariates
u=iden(Z, sep="_")      #--- identifying the test categorical covariates
D=X[,v]                 #--- matrix consisting of only training categorical covariates
E=Z[,u]                    #--- matrix consisting of only test categorical covariates

for(j in 1:ncol(D))
{
        D[,j]=as.factor(D[,j])          #--- converting training categorical column to a fact
}

for(j in 1:ncol(E))
{
        E[,j]=as.factor(E[,j])          #--- converting test categorical column to a factor
}

X1=cbind2(X[,-v],D)     #--- final training covariate matrix
Z1=cbind2(Z[,-u],E)    #--- final test covariate matrix
```

```
options(warn=-1)
library(methods)
library(glmnet)
fitg=glmnet(data.matrix(X1),as.factor(y),family="binomial",lambda=lam2)
plot(fitg,xvar="lambda",label=T)
```

```
cv.lasso=cv.glmnet(data.matrix(X1),y,family="binomial",type.measure="deviance")
plot(cv.lasso)
cv.elastic=cv.glmnet(data.matrix(X1),as.factor(y),family="binomial",alpha=0.88,type.measure="
plot(cv.elastic)
fitlasso=glmnet(data.matrix(X1),y,lambda=cv.lasso$lambda.1se,family="binomial")
fitelastic=glmnet(data.matrix(X1),as.factor(y),lambda=cv.elastic$lambda.1se,family="binomial"
p3=predict(fitelastic,newx=data.matrix(Z[,-1]),type="class")
p4=predict(fitelastic,newx=data.matrix(Z[,-1]),type="response")
mean(as.numeric(as.vector(p3))!=Z[,1])
table(p3,Z[,1])
pred_prob=predict(cv.lasso,newx=data.matrix(Z1[,-1]), type="response",s="lambda.1se")
```

```
#- Using LDA -#
library(MASS)
gfull=lda(target~.,data=X11)
plot(gfull)
fitful=predict(gfull,newdata=Z1[,-1])
mean(as.numeric(as.vector(fitful$class))!=Z[,1])
```

```
table(fitful$class,Z[,1])
fitful$posterior
```

```
pred=predict(cv.lasso,newx=data.matrix(Z1), type="response",s="lambda.1se")
pred[1:10]
cverror.lasso=cv.lasso$cvm[which(cv.lasso$lambda==cv.lasso$lambda.1se)]
cverror.lasso
```

```
library(rpart)
library(rpart.plot)
fit=rpart(target~.,data=X11,method="class",cp=0)
rpart.plot(fit)
cpseq=fit$cptable[,1]
error=function(cp)
{
        fitt=rpart(target~.,data=X11,method="class",cp=cp)
        fitt_pred=predict(fitt,newdata=Z1[,-1],type="class")
        mis=mean(as.numeric(as.vector(fitt_pred))!=Z[,1])
        return(mis)
}
cl=sapply(cpseq,error)
fit_optimum=rpart(target~.,data=X11,method="class",cp=cpseq[2])
rpart.plot(fit_optimum)
tree_pred=predict(fit_optimum,newdata=Z1[,-1], type="prob")
tree_pred2=predict(fit_optimum,newdata=Z1[,-1], type="class")
table(tree_pred2,Z[,1])
mean(as.numeric(as.vector(tree_pred2))!=Z[,1])
tree_pred
```

Bibliography

[1] Jerome Friedman, Trevor Hastie, and Rob Tibshirani of Department of Statistics of Stanford University. Regularization paths for generalized linear models via coordinate descent. 2009.

[2] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2008.

[3] Gareth James, Daniela Witten, and Trevor Hastie Robert Tibshirani. *An Introduction to Statistical Learning*. Springer.

[4] Richard A. Johnson of University of Wisconsin and Dean W. Wichern of Texas A&M University. *Applied Multivariate Statistical Analysis*. Pearson Prentice Hall.