Abstract

Telecommunication market is expanding day by day. Companies are facing a severe loss of revenue due to increasing competition hence the loss of customers. They are trying to find the reasons of losing customers by measuring customer loyalty to regain the lost customers. The customers leaving the current company and moving to another telecom company are called churn

In this report, we will discuss about the datasets, methods implemented, problems faced, results and their analysis. I have achieved good results using the Logistic Regression. I believe that we could improve the performance of the model by extracting further knowledge out of the data.

Churns can be reduced by analyzing the past history of the potential customers systematically. In the past few years, the industry has helped R programming language to emerge as one of the necessary tool for visualization, computations statistics and data science.

What is Churn is Telecom?

Churn in the terms of telecommunication industry are the customers leaving the current company and moving to another telecom company. With the increases number of churns, it has become the operators process to retain the profitable customers known as churn management.

In telecommunication industry each company provides the customers with huge incentives to lure them to switch to their services, it is one of the reasons that customer churn is a big problem in the industry nowadays

To prevent this, the company should know the reasons for which the customer decides to move on to another telecom company. It is very difficult to keep customers intact for long duration as they move to the service that suits most of their needs

Objective

A leading telecom services provider has a vast subscriber base. Our main goal is to predict churn for this telecom major.

Churn with respect to the Telecom industry, is defined as the percentage of subscribers moving from a specific service or a service provider to another in a given period of time.

Given the data of users' call and data usage patterns and demographics, the task is to:

- Build a global model that predicts churn of the entire subscriber base, where overall misclassification rate is minimized

.

STATE OF THE ART

- Data volume has been rising at an incredible step over the last two decades due to progressions in information knowledge.
- The toughest difficulty faced by the telecom industry to identify customer churn.
- Retention of old clients is always desirable option to the company i.e, prevent them from churn to another Service Provider.
- Moreover, Attracting new clients costs almost 5-6 times more than retentive the old clients
- Attracting a new client comprises new recruits of manpower, cost of publicity & discounts.
- A loyal client, who has been with the business for quite the long time, tends to produce higher profits, such clients also cost less to keep.
- A minor step towards retentive a current customer can lead to an important growth in revenues & profits.
- Therefore there is a require for Predictive Models to classify customers who are about to churn & their reason for churn to evade losses to industry of telecom, the Model should be recognised to classify the reasons to churn & the enhancements required to recollect customers.

DIFFERENT MODELS BUILD IN THE PROJECT

A. Logistic Regression
B. Deep Learning
C. Decision Trees (rpart)
D. Random Forest
E. Support Vector Machine

A. Logistic Regression

It is a form of regression that allows the prediction of discrete variables by a mic of continuous and discrete predictors.

It addresses the same questions that discriminant function analysis and multiple regression do buy with no distributional assumptions on the predictors. The predictors do not have to be normally distributed, linearly related or have equal variance in each group.

Types of logistic regression

i. Binary Logistic Regression
ii. Multinomial Logistic Regression

i. Binary Logistic Regression:

It is used when the dependent variable is dichotomous.

ii. Model Based Technique:

It is used when the dependent or outcomes variable has more than two categories.

In the logistic regression the outcome variable is binary and the purpose of the analysis is to assess the effects of multiple explanatory variables, which can be numeric and/or categorical on the outcome variable.

B. Deep Learning

Deep learning is characterized as a class of machine learning algorithms that:

- Use a cascade of many layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input. The algorithms may be supervised or unsupervised and applications include pattern analysis (unsupervised) and classification (supervised).
- are based on the (unsupervised) learning of multiple levels of features or representations of the data. Higher level features are derived from lower level features to form a hierarchical representation.
- are part of the broader machine learning field of learning representations of data.
- learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.

These definitions have in common (1) multiple layers of nonlinear processing units and (2) the supervised or unsupervised learning of feature representations in each layer, with the layers forming a hierarchy from low-level to high-level features

C. Decision Trees

Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems.

It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.

Types of Decision Trees

i. Categorical Variable Decision Tree
ii. Continuous Variable Decision Tree

i. Categorical Variable Decision Tree

Decision Tree which has categorical target variable then it called as categorical variable decision tree. Example:- In above scenario of student problem, where the target variable was “Student will play cricket or not” i.e. YES or NO.

ii. **Continuous Variable Decision Tree**
Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree.

Cons of Decision Tree:

- Easy to understand
- Useful in data exploration
- Less data cleaning required
- Data type is not a constant
- Non parametric method

D. Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set

Random forests differ in only one way from this general scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated.

E. Support Vector Machine

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification and regression challenges. However, it is mostly used in classification problems.

In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well (look at the below snapshot).

Pros and Cons associated with SVM

- **Pros:**
  - It works really well with clear margin of separation
  - It is effective in high dimensional spaces.
  - It is effective in cases where number of dimensions is greater than the number of samples.
  - It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

- **Cons:**
  - It doesn't perform well, when we have large data set because the required training time is higher
  - It also doesn't perform very well, when the data set has more noise i.e. target classes are overlapping
  - SVM doesn't directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is related SVC method of Python scikit-learn library.

A. Problem Statement

METHOD

The data provided is related to Churn of some telecom services provider. And also most of the data provided do not have data dictionary

Consumers today go through a complex decision making process before subscribing to any one of the numerous Telecom service options. Since the services provided by the Telecom vendors are not highly differentiated and number portability is commonplace, customer loyalty becomes an issue. Hence, it is becoming increasingly important for telecommunications companies to proactively identify customers that have a tendency to unsubscribe and take preventive measures to retain such customers.
Our main goal is to **predict churn for this telecom** data.

B. Approach

Following are the steps to be followed for churn prediction:

- Standardizing the data
- Removing attributes that are highly correlated
- Plotting the outliers & eliminating them
- Splitting the data in train and test
- Use Auto encoders to extract the customized features

Now we have all the attributes that are important in building churn prediction models

Error metrics that are considered are Accuracy,Precession and Recall

	Predicted as +Ve	Predicted as –Ve
Actual +ve	T rue +ve	False –ve
Actual –ve	False +ve	True –ve

DATA

Below is the given Churn Data

	A	B	C	D	E	F	G	H	I	J	K	L
1	s6.new.rev.p2.m2	s1.new.rev.m1	s3.og.rev.4db.p5	s3.new.rev.4db.p5	s4.usg.ins.p2	s4.og.unq.any.p2	s2.rch.val.p6	s1.og.rev.all.m1	s8.new.rev.p6	s4.loc.ic.ins.p1	s8.mbl.p2	s2.rch.val.l67
2	-0.76	88.0482	3.10660413	3.754955148	4	14	39.29	57.32	-0.17	1	-0.72	39.44
3	-0.98	67.5039	3.094573675	5.550864626	1	2	21.67	38.7	-0.32	3	-0.08	18.89
4	-0.98	33.9248	2.324016435	2.438114214	2	3	30	15.32	-0.05	3	-0.09	29.5
5	-0.92	82.678	2.630748526	2.858961459	2	3	50	51.956	-0.18	4	1.83	46.67
6	-0.97	96.8379	2.674316446	2.912396571	3	2	22.5	66.886	0.01	4	-0.04	37.2
7	0.04	968.7667	14.71721599	23.58982727	7	62	177.71	882.935	-0.79	4	11.77	162.48
8	2.41	173.0715	5.324949542	7.383014538	4	39	44.83	122.2	0.13	4	2.91	49.42
9	0.65	82.7458	4.109535759	5.398088606	7	18	99.4	76.99	0.28	4	15.28	85.29
10	-1	787.2101	22.59887927	23.57469623	0	0	69.09	737.851	-0.32	0	0	71.2
11	-0.48	148.9607	2.650347171	2.982616274	7	23	51.2	98.9144	0.03	4	-4.16	52.14
12	-0.14	303.6129	3.185628227	4.042776065	7	41	65	289.212	0.15	4	-8.92	63.64
13	0.65	724.7486	12.12604817	12.0991113	7	98	255	716.115	0.57	4	-3.34	193.33
14	-0.98	183.7341	4.750870565	4.98358564	2	4	80	156.36	-0.28	1	-0.08	61.25
15	1.79	105.0521	3.553805711	3.571837328	6	19	29	97.366	0.2	3	-2.08	30.56
16	0.86	129.0361	5.070458106	6.769899414	7	74	34.9	96.397	0.38	4	6.3	35.31
17	0.07	0	2.203046811	3.082171859	2	15	98.33	0	-0.14	4	7.39	105.29
18	-0.33	451.4854	4.33093951	5.389363649	4	10	136	421.791	-0.14	3	-1.7	139.25
19	-0.34	241.0346	2.931047861	7.764599109	7	46	79.8	192.084	-0.33	4	-5.77	75.44
20	-0.26	366.0499	4.79830935	6.820876252	7	34	70	299.205	-0.19	4	7.67	64.73
21	-0.63	10.178	3.389860576	4.985131893	7	10	45	10.178	0.04	2	-1.74	45
22	-0.55	161.3823	10.45552112	10.45552112	7	13	199.33	158.5	0.46	1	-4.44	171.4

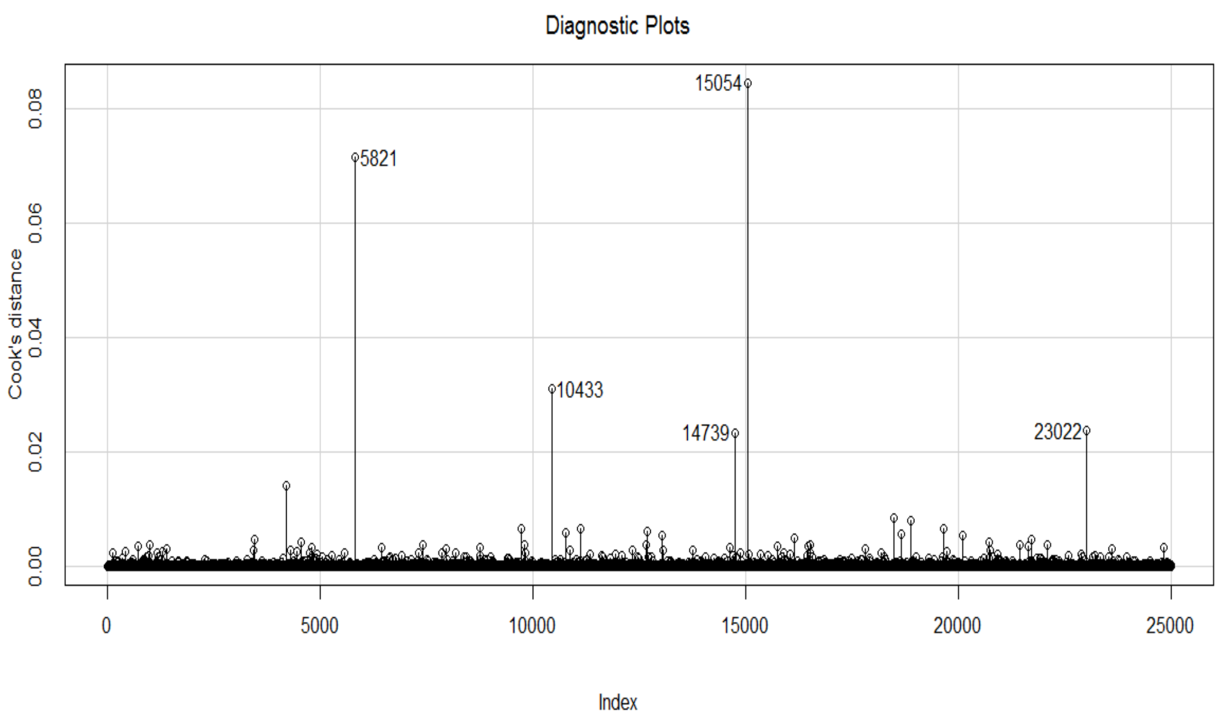
Summary of the given churn data

```
> summary(ChurnData)
 s6.new.rev.p2.m2   s1.new.rev.m1    s3.og.rev.4db.p5  s3.new.rev.4db.p5   s4.usg.ins.p2   s4.og.unq.any.p2 s2.rch.val.p6
 Min.   : -1.0000   Min.   :    0.0  Min.   :  0.000   Min.   :  0.00083   Min.   :0.00    Min.   :  0.00   Min.   :   0.00
 1st Qu.: -0.5800   1st Qu.: 101.6   1st Qu.:  2.367   1st Qu.:  3.31883   1st Qu.:5.00    1st Qu.:  9.00   1st Qu.:  33.00
 Median : -0.1700   Median : 204.9   Median :  3.730   Median :  5.23127   Median :7.00    Median : 21.00   Median :  52.26
 Mean   : -0.0037   Mean   : 281.1   Mean   :  4.890   Mean   :  7.07019   Mean   :5.46    Mean   : 28.53   Mean   :  72.21
 3rd Qu.:  0.2800   3rd Qu.: 370.7   3rd Qu.:  5.993   3rd Qu.:  8.39574   3rd Qu.:7.00    3rd Qu.: 39.00   3rd Qu.:  89.85
 Max.   :316.8600   Max.   :5702.9   Max.   :153.222   Max.   :170.20044   Max.   :7.00    Max.   :622.00   Max.   :2249.00
 s1.og.rev.all.m1  s8.new.rev.p6     s4.loc.ic.ins.p1   s8.mbl.p2             s2.rch.val.l67    s7.s4.day.no.mou.p2.p4
 Min.   :   0.00   Min.   :-5.09000  Min.   :0.000   Min.   :-153.75000   Min.   :   0.00   Min.   : 0.0000
 1st Qu.:  74.42   1st Qu.:-0.16000  1st Qu.:3.000   1st Qu.:  -2.56000   1st Qu.:  33.50   1st Qu.: 0.1000
 Median : 151.17   Median :-0.02000  Median :4.000   Median :  -0.08000   Median :  51.67   Median : 0.3636
 Mean   : 218.52   Mean   :-0.02833  Mean   :3.255   Mean   :  -0.07391   Mean   :  71.35   Mean   :28.4815
 3rd Qu.: 284.26   3rd Qu.: 0.11000  3rd Qu.:4.000   3rd Qu.:   1.04000   3rd Qu.:  87.58   3rd Qu.:99.0000
 Max.   :3767.57   Max.   : 5.00000  Max.   :4.000   Max.   : 183.35000   Max.   :1000.00   Max.   :99.0000
 s3.new.rev.p3    s7.s5.s4.day.nomou.p4  s8.og.rev.p3        s8.ic.mou.all.p3       target          s7.new.rev.p2.p6
 Min.   :  0.00   Min.   :0.0000         Min.   :-31.67000   Min.   :-24.29000   Min.   :0.0000   Min.   :0.0000
 1st Qu.:  4.43   1st Qu.:0.0000         1st Qu.: -0.33000   1st Qu.: -0.29000   1st Qu.:0.0000   1st Qu.:0.0900
 Median :  7.82   Median :0.0000         Median : -0.03000   Median : -0.02000   Median :0.0000   Median :0.1600
 Mean   : 10.96   Mean   :0.2842         Mean   : -0.07492   Mean   : -0.05156   Mean   :0.3167   Mean   :0.1666
 3rd Qu.: 13.06   3rd Qu.:0.5000         3rd Qu.:  0.20000   3rd Qu.:  0.21000   3rd Qu.:1.0000   3rd Qu.:0.2300
 Max.   :395.12   Max.   :1.0000         Max.   :  7.88000   Max.   : 35.92000   Max.   :1.0000   Max.   :0.9900
 s6.rtd.mou.p2.m2   s7.rtd.mou.p2.p6 s1.new.rev.p2     s1.new.rev.p1     s1.og.hom.mou.p1    s7.rev.p2.p6    s1.og.hom.rev.p2
 Min.   :-1.00000   Min.   :0.0000   Min.   :   0.00   Min.   :  0.000   Min.   :   0.000   Min.   :-4.31   Min.   :   0.00
 1st Qu.:-0.54000   1st Qu.:0.1000   1st Qu.:  18.61   1st Qu.:  7.701   1st Qu.:   5.516   1st Qu.: 0.09   1st Qu.:  12.82
 Median :-0.15000   Median :0.1600   Median :  46.03   Median : 23.917   Median :  21.166   Median : 0.16   Median :  34.20
 Mean   :-0.01592   Mean   :0.1683   Mean   :  70.22   Mean   : 39.056   Mean   :  45.029   Mean   : 0.17   Mean   :  52.51
 3rd Qu.: 0.27000   3rd Qu.:0.2300   3rd Qu.:  92.12   3rd Qu.: 51.593   3rd Qu.:  52.954   3rd Qu.: 0.24   3rd Qu.:  69.38
 Max.   :34.62000   Max.   :0.8900   Max.   :1683.07   Max.   :992.370   Max.   :2924.100   Max.   : 9.56   Max.   :1206.66
 s1.rtd.mou.p1     s1.og.rev.all.p1 s1.og.mou.all.p1   s3.og.rev.all.p1  s7.new.rev.p3.p6   ds.usg.p6        snd.dec.p2
 Min.   :   0.00   Min.   :  0.00   Min.   :   0.000   Min.   :  0.000   Min.   :0.0000   Min.   : 0.000   Min.   :-0.85000
 1st Qu.:   6.80   1st Qu.:  6.01   1st Qu.:   6.283   1st Qu.:  1.340   1st Qu.:0.2900   1st Qu.: 0.000   1st Qu.:-0.64000
 Median :  23.09   Median : 19.06   Median :  22.083   Median :  3.540   Median :0.3900   Median : 0.000   Median :-0.30000
 Mean   :  47.51   Mean   : 30.59   Mean   :  46.066   Mean   :  5.431   Mean   :0.3803   Mean   : 1.349   Mean   :-0.01051
 3rd Qu.:  55.98   3rd Qu.: 40.53   3rd Qu.:  54.166   3rd Qu.:  7.070   3rd Qu.:0.4800   3rd Qu.: 0.000   3rd Qu.: 0.26000
 Max.   :2924.10   Max.   :917.86   Max.   :2924.100   Max.   :213.800   Max.   :1.0000   Max.   :35.000   Max.   :21.95000
 s3.og.mou.all.p1  ds.og.usg.p4     s1.og.mou.all.p2   s8.og.rev.p6       s1.og.hom.mou.p2  s5.og.rev.all.p1 s1.og.rev.all.p2
 Min.   :  0.00    Min.   : 0.000   Min.   :   0.00    Min.   :-3.37000   Min.   :   0.00   Min.   :  0.00   Min.   :   0.00
```

Correlated attributes using findCorrelation()

```
> highlycorrelated
 [1]  10  13  15  43  49  50  51  53  54  58  61  62  66  68  69  71  72  74  76  77  78  80
[23]  84  86  92  93  94  95  96  97  98  99 101 102 103 105 108 109  11   9  70  16  45  81
[45]  17
```

Outliers



Finding important attributes using AutoEncoders

```
> names(features)
 [1] "DF.C1"  "DF.C2"  "DF.C3"  "DF.C4"  "DF.C5"  "DF.C6"  "DF.C7"  "DF.C8"
 [9] "DF.C9"  "DF.C10" "DF.C11" "DF.C12" "DF.C13" "DF.C14" "DF.C15" "DF.C16"
[17] "DF.C17" "DF.C18" "DF.C19" "DF.C20" "DF.C21" "DF.C22" "DF.C23" "DF.C24"
[25] "DF.C25" "DF.C26" "DF.C27" "DF.C28" "DF.C29" "DF.C30"
```

Logistic Regression Result

```
> #Confusion Metrics
> LR_conf_Matrix=table(test$Target, pred_class)
>
> #Error Metrics
> LR_acc = sum(diag(LR_conf_Matrix))/sum(LR_conf_Matrix)*100
> LR_prec = LR_conf_Matrix[2,2]/sum(LR_conf_Matrix[,2])*100
> LR_recall= LR_conf_Matrix[2,2]/sum(LR_conf_Matrix[2,])*100
>
> #Storing the Error Metrics values into a vector
> LogisticReg_Results =c("accuracy"=LR_acc,"precision"=LR_prec, "recall"=LR_recall)
> LogisticReg_Results
 accuracy precision    recall
 79.38667  73.56383  56.86678
```

Deep Learning Result

```
> #Confusion Matrix
> conf_Matrix=table(test$Target, pred$predict)
>
> #Error Metrics
> dl_acc = sum(diag(conf_Matrix))/sum(conf_Matrix)*100
> dl_prec = conf_Matrix[2,2]/sum(conf_Matrix[,2])*100
> dl_recall= conf_Matrix[2,2]/sum(conf_Matrix[2,])*100
>
> #Storing the Error Metrics values into a vector
> DeepLearning_Results =c("accuracy"=dl_acc,"precision"=dl_prec, "recall"=dl_recall)
> DeepLearning_Results
 accuracy precision    recall
 79.44000  70.43159  63.07566
```

Decision Tree (rpart) Result

```
> #Confusion matrix for Test data Predictions
> DT_conf_Matrix=table(FCdata_test$Target, dt_model_test_pred)
>
> #Error Metrics
> DT_acc = sum(diag(DT_conf_Matrix))/sum(DT_conf_Matrix)*100
> DT_prec = DT_conf_Matrix[2,2]/sum(DT_conf_Matrix[,2])*100
> DT_recall= DT_conf_Matrix[2,2]/sum(DT_conf_Matrix[2,])*100
>
> #Storing the Error Metrics values into a vector
> DecissionTree_Results =c("accuracy"=DT_acc,"precision"=DT_prec, "recall"=DT_recall)
> DecissionTree_Results
 accuracy precision    recall
 78.12000  69.09589  53.93499
```

Random Forest Result

```
> #Confusion matrix for Test data Predictions
> RF_conf_Matrix=table(FCdata_test$Target, rf_model_test_pred)
>
> #Error Metrics
> RF_acc = sum(diag(RF_conf_Matrix))/sum(RF_conf_Matrix)*100
> RF_prec = RF_conf_Matrix[2,2]/sum(RF_conf_Matrix[,2])*100
> RF_prec= RF_conf_Matrix[2,2]/sum(RF_conf_Matrix[2,])*100
>
> #Storing the Error Metrics values into a vector
> RandomForest_Results =c("accuracy"=RF_acc,"precision"=RF_prec, "recall"=RF_prec)
> RandomForest_Results
 accuracy precision    recall
 76.78667  54.49102  54.49102
```

SVM Results

```
> #Confusion Metrics for the Test data prediction
> svm_conf_Matrix=table(FCdata_test$Target, svm_model_test_pred)
>
> #Error Metrics
> svm_acc = sum(diag(svm_conf_Matrix))/sum(svm_conf_Matrix)*100
> svm_prec = svm_conf_Matrix[2,2]/sum(svm_conf_Matrix[,2])*100
> svm_prec= svm_conf_Matrix[2,2]/sum(svm_conf_Matrix[2,])*100
>
> #Storing the Error Metrics values into a vector
> SVM_Results =c("accuracy"=svm_acc,"precision"=svm_prec, "recall"=svm_prec)
> SVM_Results
 accuracy precision    recall
 72.80000  46.15056  46.15056
```

RESULT

	Accuracy	Precision	Recall
Logistic Regression	79.39	73.56	56.87
Deep Learning	78.76	76.70	49.54
DecissionTree (rpart)	78.12	69.10	53.93
RandomForest	76.79	54.49	54.49
SVM	72.80	46.15	46.15

As we can observe from the above results that **Logistic Regression** surpasses both Decision Trees and Support Vector Machine and it is also easy to construct. Hence we would recommend Logistic Regression model for predicting the churn.

Selecting the right combination of attributes and fixing the proper threshold values may produce more accurate results.

SOURCE CODE

```
# Clear complete Work Space
rm(list=ls(all=TRUE))

# Setting the working directory Path
setwd("D:/DA/CSE9099_CPEE Project/data")

#reading data from the csv file
ChurnData = read.csv("Data.csv",header = T,sep = ",")

#To view the column names
names(ChurnData)

#checking for the missinng values
sum(is.na(ChurnData))

#View structure and summary of the data
str(ChurnData)
summary(ChurnData)

#converting target variable to factor
ChurnData$target = as.factor(ChurnData$target)
str(ChurnData$target)

#removing target attribute
S_ChurnData = ChurnData[,-c(18)]

#Standardizing the data Using range method
S_ChurnData = decostand(S_ChurnData,"range")

#install.packages("car")
library(car)
#install.packages("MASS")
```

```
library(MASS)

library(caret)
#install.packages("mlbench")
library(mlbench)

# Finding the highly correlated columns
highlycorrelated = findCorrelation(S_ChurnData)
highlycorrelated

# Removing the high corelated columns from the data
Final_ChurnData = S_ChurnData[,-c(highlycorrelated)] # Reduced
features to 65 from 110

#Adding back the target column to the 'Final_ChurnData' dataframe
Final_ChurnData$Target <- ChurnData$target
names (Final_ChurnData)

# Finding the outliers
Churn_LogReg <- glm(Target~.,data = Final_ChurnData,family =
binomial())
influenceIndexPlot(Churn_LogReg,vars = c("cook"),id.n = 5)

#Removing the outliers in the final data
Final_ChurnData<- Final_ChurnData[-
c(5821,10433,14739,15054,23022)]

# Splitting the data into train & test with 70% and 30%
rows=seq(1,nrow(Final_ChurnData),1)
set.seed(123)
trainRows=sample(rows,(70*nrow(Final_ChurnData))/100)
train = Final_ChurnData[trainRows,]
test = Final_ChurnData[-trainRows,]

############################## Logistic Regression
##############################
```

```
#Creating Logistic regression on the train dataset
LogReg1 = glm(Target ~ ., family = binomial(),data=train)
summary(LogReg1)

#Predicting the test dataframe using logistic regreesion model created
predictTest = predict(LogReg1, type="response", newdata=test)

#Converting the value into binary
pred_class = factor(ifelse(predictTest> 0.5, 1, 0))
pred_class

#library(caret)
#Confusion Metrics
LR_conf_Matrix=table(test$Target, pred_class)

#Error Metrics
LR_acc = sum(diag(LR_conf_Matrix))/sum(LR_conf_Matrix)*100
LR_prec = LR_conf_Matrix[2,2]/sum(LR_conf_Matrix[,2])*100
LR_recall= LR_conf_Matrix[2,2]/sum(LR_conf_Matrix[2,])*100

#Storing the Error Metrics values into a vector
LogisticReg_Results =c("accuracy"=LR_acc,"precision"=LR_prec,
"recall"=LR_recall)
LogisticReg_Results
##################### Features Extraction using AEC
##########################

library(h2o)

# Initiate h2o process - can assign ip/port/max_mem_size(ram size)/
# nthreads(no. of processor cores; 2-2core;-1 -all cores available)
localh2o <- h2o.init(ip='localhost', port = 54321, max_mem_size =
'1g',nthreads = 1)

#Converting R object to an H2O Object
```

```
Final_churn.hex <- as.h2o(localh2o, object = Final_ChurnData, key =
"Final_churn.hex")

#To extract features using autoencoder method
aec <- h2o.deeplearning(x = setdiff(colnames(Final_churn.hex),
"Target"),
                        y = "Target", data = Final_churn.hex,
                        autoencoder = T, activation = "RectifierWithDropout",
                        classification = T, hidden = c(30),
                        epochs = 100, l1 = 0.01)

#Converting R object to an H2O Object
train.hex <- as.h2o(localh2o, object = train, key = "train.hex")
test.hex <- as.h2o(localh2o, object = test, key = "test.hex")

#################### DeepLearning Model
####################################

#DeepLearning model implementation using the AEC features
dl_model = h2o.deeplearning(x = setdiff(colnames(train.hex),
"Target"),
                           y = "Target",
                           data = train.hex,
                           # activation =  "Tanh",
                           hidden = c(5, 10, 10),
                           activation = "RectifierWithDropout",
                           input_dropout_ratio = 0.1,
                           epochs = 100,seed=123)

#Prediction on test data
prediction = h2o.predict(dl_model, newdata = test.hex)

#Convert prediction from h2o object to R object/dataframe
pred = as.data.frame(prediction)

#Confusion Matrix
```

```
conf_Matrix=table(test$Target, pred$predict)

#Error Metrics
dl_acc = sum(diag(conf_Matrix))/sum(conf_Matrix)*100
dl_prec = conf_Matrix[2,2]/sum(conf_Matrix[,2])*100
dl_recall= conf_Matrix[2,2]/sum(conf_Matrix[2,])*100

#Storing the Error Metrics values into a vector
DeepLearning_Results =c("accuracy"=dl_acc,"precision"=dl_prec,
"recall"=dl_recall)
DeepLearning_Results
##################### Converting AEC extracted features into R
##############################

# Converting the AEC extracted features into R dataframe
features =
as.data.frame.H2OParsedData(h2o.deepfeatures(Final_churn.hex[,-66],
model = aec))

# Adding the Target column to the extracted features
Featured_ChurnData = cbind(features,ChurnData$target)

# Renaming the 'ChurnData$target' column name as 'Target'
names(Featured_ChurnData)[31] = "Target"

# Split the 'Featured_Churndata' into train and test with 70% & 30%
set.seed(1234)
trainrows = sample(nrow(Featured_ChurnData),0.7 *
nrow(Featured_ChurnData))

FCdata_train = Featured_ChurnData[trainrows,]
FCdata_test = Featured_ChurnData[-trainrows,]

####################### Decission Tree - RPART
################################
```

```
library(rpart)

dt_model =
rpart(Featured_ChurnData$Target~.,Featured_ChurnData,method =
"class")
summary(dt_model)

#Predicting on train data
dt_model_train_pred = predict(dt_model,newdata =
FCdata_train,type="class")

#Predicting on train data
dt_model_test_pred = predict(dt_model,newdata =
FCdata_test,type="class")

#Confusion matrix for Test data Predictions
DT_conf_Matrix=table(FCdata_test$Target, dt_model_test_pred)

#Error Metrics
DT_acc = sum(diag(DT_conf_Matrix))/sum(DT_conf_Matrix)*100
DT_prec = DT_conf_Matrix[2,2]/sum(DT_conf_Matrix[,2])*100
DT_recall= DT_conf_Matrix[2,2]/sum(DT_conf_Matrix[2,])*100

#Storing the Error Metrics values into a vector
DecissionTree_Results =c("accuracy"=DT_acc,"precision"=DT_prec,
"recall"=DT_recall)
DecissionTree_Results
############################## Random Forest
######################################

set.seed(12345)
library(randomForest)

rf_model = randomForest(FCdata_train$Target~.,data=FCdata_train,
keep.forest=TRUE, ntree=10)
summary(rf_model)
```

```
# Predict on Train data
rf_model_train_pred =
predict(rf_model,FCdata_train,type="response")

# Predicton Test Data
rf_model_test_pred <-predict(rf_model,FCdata_test,type="response")

#Confusion matrix for Test data Predictions
RF_conf_Matrix=table(FCdata_test$Target, rf_model_test_pred)

#Error Metrics
RF_acc = sum(diag(RF_conf_Matrix))/sum(RF_conf_Matrix)*100
RF_prec = RF_conf_Matrix[2,2]/sum(RF_conf_Matrix[,2])*100
RF_prec= RF_conf_Matrix[2,2]/sum(RF_conf_Matrix[2,])*100

#Storing the Error Metrics values into a vector
RandomForest_Results =c("accuracy"=RF_acc,"precision"=RF_prec,
"recall"=RF_prec)
RandomForest_Results
######################################## SVM
########################################
library(e1071)
#SVM on the data which has all columns after removing the highly
corelated columns
#svm_model2 <- svm (Target ~.,data = train, kernel = "radial", cost =
10, gamma=0.1)

#Predict on the test data
#svm_model2_test_pred = predict(svm_model2, test)

#Confusion Metrics on the test data
#svm_conf_Matrix2=table(FCdata_test$Target,
svm_model2_test_pred)

#Error Metrics
```

```
#svm_acc = sum(diag(svm_conf_Matrix2))/
sum(svm_conf_Matrix2)*100
#svm_prec = svm_conf_Matrix2[2,2]/sum(svm_conf_Matrix2[,
2])*100
#svm_prec= svm_conf_Matrix2[2,2]/sum(svm_conf_Matrix2[2,])*100

#As the accuracy is low, trying to build the model on the AEC
extracted features
#SVM model on the AEC extracted features
svm_model <- svm (Target ~.,data = FCdata_train, kernel = "radial",
cost = 10, gamma=0.1)

#Predicting the model on teh test data
svm_model_test_pred = predict(svm_model, FCdata_test)

#Confusion Metrics for the Test data prediction
svm_conf_Matrix=table(FCdata_test$Target, svm_model_test_pred)

#Error Metrics
svm_acc = sum(diag(svm_conf_Matrix))/sum(svm_conf_Matrix)*100
svm_prec = svm_conf_Matrix[2,2]/sum(svm_conf_Matrix[,2])*100
svm_prec= svm_conf_Matrix[2,2]/sum(svm_conf_Matrix[2,])*100

#Storing the Error Metrics values into a vector
SVM_Results =c("accuracy"=svm_acc,"precision"=svm_prec,
"recall"=svm_prec)
SVM_Results
############################## Final Result
########################################

#Final Result displaying the Error Metrics for all the above models
Final_Result = t(data.frame(LogisticReg_Results,
                            DeepLearning_Results,
                            DecissionTree_Results,
                            RandomForest_Results,
                            SVM_Results))
```

```
View(Final_Result)
```