Churn Reduction

Mohana Krishna Polamuri

28 May 2018

Contents

Chapter 1

Introduction

1.1 Problem Statement

Churn (loss of customers to competition) is a problem for companies because it is more expensive to acquire a new customer than to keep your existing one from leaving. We are provided with a data set in which customer telephone service and call details. This problem statement is targeted at enabling churn reduction using analytics concepts.

1.2 Data

The objective of this Case is to predict customer behaviour. We are provided with a public dataset that has customer usage pattern and if the customer has moved or not. We have to develop an algorithm to predict the churn score based on usage pattern.

Table 1.1: Churn Reduction Train data (Columns: 1-6)

State	Account length	Area code	Phone number	International plan	Voice mail plan
KS	128	415	382-4657	No	Yes
OH	107	415	371-7191	No	Yes
NJ	137	415	358-1921	No	No
OH	84	408	375-9999	Yes	No
OK	75	415	330-6626	Yes	No
AL	118	510	391-8027	Yes	No

Table 1.2: Churn Reduction Train data (Columns: 7-12)

number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls
25	265.1	110	45.07	197.4	99
26	161.6	123	27.47	195.5	103
0	243.4	114	41.38	121.2	110
0	299.4	71	50.9	61.9	88
0	166.7	113	28.34	148.3	122
0	223.4	98	37.98	220.6	101

Table 1.3: Churn Reduction Train data (Columns: 13-18)

total eve charge	total night minutes	total night calls	total night charge	total intl minutes	total intl calls
16.78	244.7	91	11.01	10	3
16.62	254.4	103	11.45	13.7	3
10.3	162.6	104	7.32	12.2	5
5.26	196.9	89	8.86	6.6	7
12.61	186.9	121	8.41	10.1	3
18.75	203.9	118	9.18	6.3	6

Table 1.3: Churn Reduction Train data (Columns: 19-21)

total intl charge	Number customer service calls	Churn
2.7	1	False.
3.7	1	False.
3.29	0	False.
1.78	2	False.
2.73	3	False.
1.7	0	False.

There are 17 predictor variable and are as follows.

- account length
- international plan
- voicemail plan
- number of voicemail messages
- total day minutes
- total day calls
- total day charge
- total evening minutes
- total evening calls
- total evening charge
- total night minutes
- total night calls
- total night charge
- total international minutes used
- total international calls made
- total international charge
- number of customer service calls made

Our target variable is churn of which we have to say whether the customer will move or not.

Chapter 2

Methodology

2.1 Pre Processing

Any predictive modeling requires that we look at the data before we start modeling. However, in data mining terms *looking at data* refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as **Exploratory Data Analysis**. For our data we apply pre processing techniques that we necessary.
Our pre processing involves following steps:

1. Missing value analysis
2. Feature selection
   2.1 Correlation Analysis
   2.2 Chi square test of Independence
3. Normalization

2.1.1 Missing value analysis

Missing values occur when no data value is stored for the variable in an observation. Missing values are a common occurrence, and you need to have a strategy for treating them. A missing value can signify a number of different things in your data. Perhaps the data was not available or not applicable or the event did not happen. It could be that the person who entered the data did not know the right value, or missed filling in. Typically, ignore the missing values, or exclude any records containing missing values, or replace missing values with the mean, or infer missing values from existing values. We check for missing values in our data. Fortunately there are no missing values in the data that is presented.

2.1.2 Feature Selection

Variable selection is an important aspect of model building. It helps in building predictive models free from correlated variables, biases and unwanted noise. It helps in selecting a subset of relevant features (variables, predictors) for use in model construction and subset of a learning algorithm's input variables upon which it should focus attention, while ignoring the rest. Let's first see how our target variable is behaving with categorical variables in dataset.

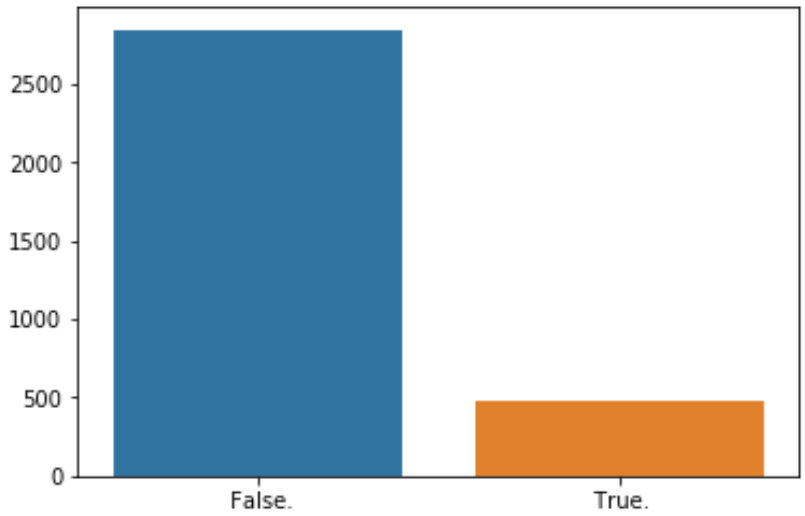


Fig 1 Churn data distribution

Distribution of Churn with each categorical variable in data

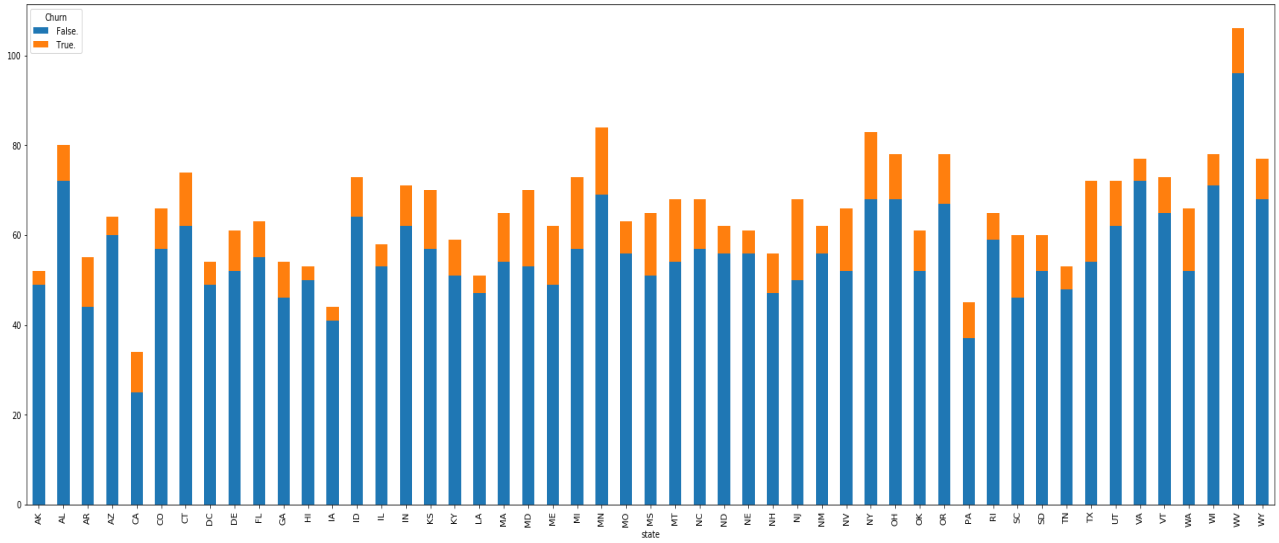


Fig 2 Churn vs State

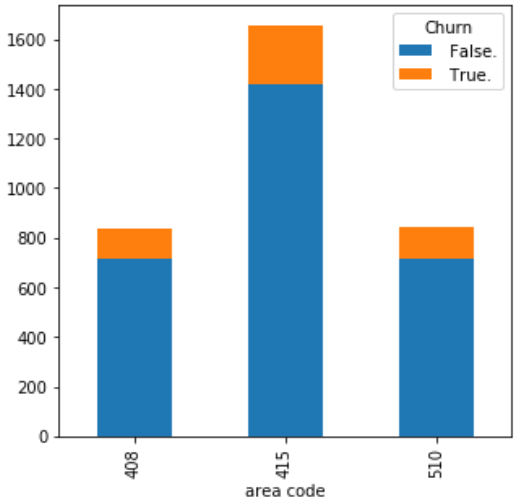


Fig 3 Area code vs Churn

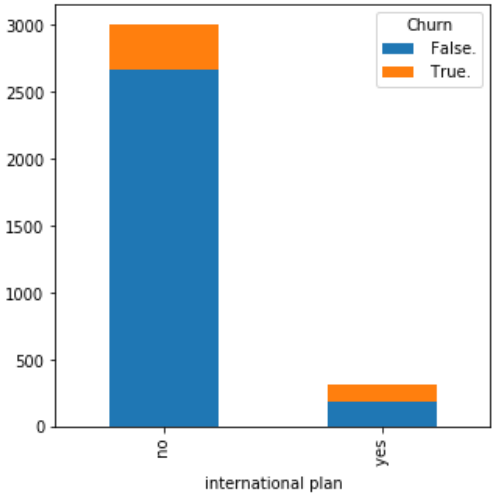


Fig 4 International plan vs Churn

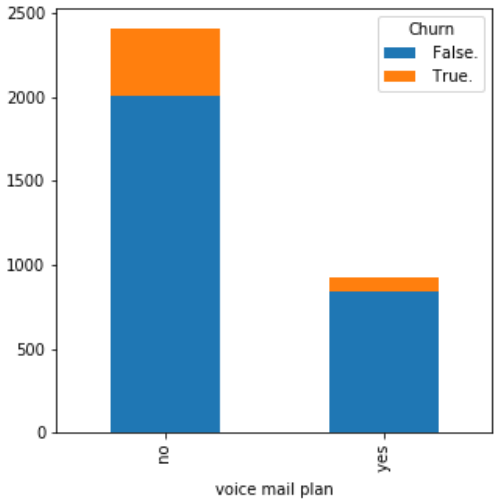


Fig 5 voice mail plan vs Churn

We will be assigning levels to our categorical data so that I will easy apply model and do pre processing techniques. Code for converting can be seen in appendix.

Correlation analysis

A correlation matrix is a table showing correlation coefficients between sets of variables. Each random variable (Xi) in the table is correlated with each of the other values in the table (Xj). This allows you to see which pairs have the highest correlation. Checking at the correlation of each numerical variable has given below result.

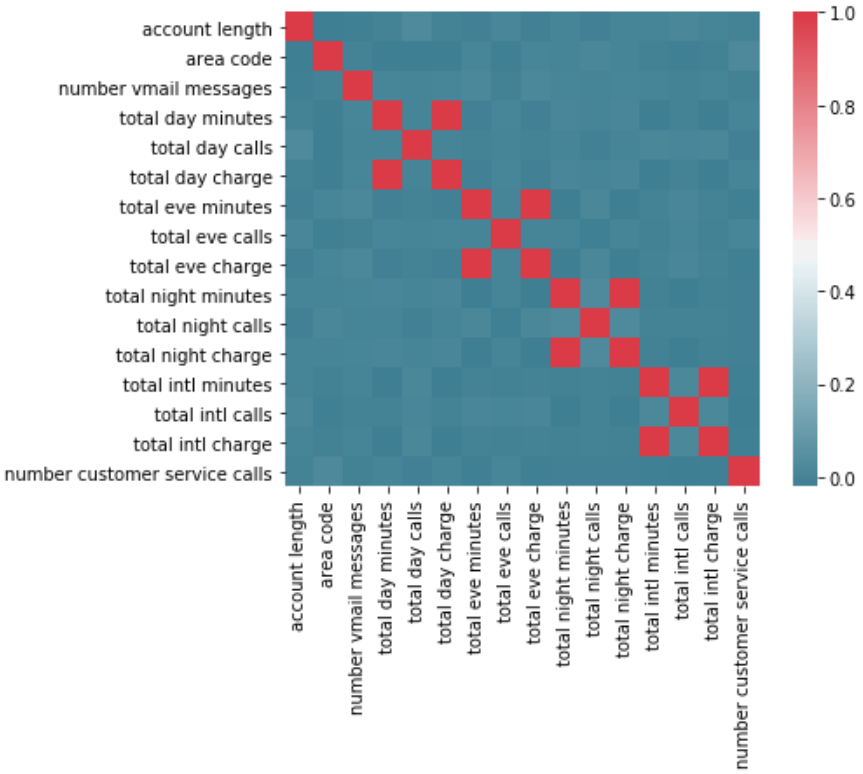


Fig 6 Correlation Matrix

Our correlation matrix shows some interesting results as follows

1. Total day minutes and total day charge are very highly correlated.
2. Total eve minutes and total eve charge are very highly correlated.
3. Total night minutes and total night charge are very highly correlated.
4. Total intl minutes and total intl charge are very highly correlated.

We can now remove one of the highly correlated variable so that our model can perform well with much accuracy.

Chi square test of Independence

The Chi-Square test of independence is used to determine if there is a significant relationship between two categorical variables. The frequency of each category for one variable is compared across the categories of the second variable. It is used to determine whether there is a significant association between the two variables. Uses contingency table for better representation. We conduct chi square test of independence for each of the categorical variable with our target variable that is churn so that we will remove the variable that is independent with target variable. Scores of chi square test of independence of each categorical variable is as follows

"**state**"

Pearson's Chi-squared test

data: table(factor_data$Churn, factor_data[, i])
X-squared = 83.044, df = 50, p-value = 0.002296

"**phone number**"

Pearson's Chi-squared test

data: table(factor_data$Churn, factor_data[, i])
X-squared = 3333, df = 3332, p-value = 0.4919

"**international plan**"

Pearson's Chi-squared test with Yates' continuity correction

data: table(factor_data$Churn, factor_data[, i])
X-squared = 222.57, df = 1, p-value < 2.2e-16

"**voice mail plan**"

Pearson's Chi-squared test with Yates' continuity correction

data: table(factor_data$Churn, factor_data[, i])
X-squared = 34.132, df = 1, p-value = 5.151e-09

If the p value of the categorical variable is less than 0.05 then we will consider that variable that is the target variable is dependent on the categorical variable.

Therefore from both the correlation analysis and chi square test of independence we got some variable which we shouldn't consider for further processing. The variables that could be deleted are as follows

Numerical: total day minutes, total eve minutes, total night minutes, total intl minutes
Categorical: phone number.

2.1.3 Feature Scaling

Feature scaling is a method used to standardize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing steps. If training an algorithm using different features and some of them are off the scale in their magnitude, then the results might be dominated by them. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance. We use normalization here for feature scaling.

Normalization also called Min-Max scaling. It is the process of reducing unwanted variation either within or between variables. Normalization brings all of the variables into proportion with one another. It transforms data into a range between 0 and 1. We have to see the variables that are scattered highly and apply normalization. We normalize the following variables in our data so that we can process to the modeling phase. Normality check for variables is in appendix

Variables
account length, area code, number vmail messages, total day calls, total day charge, total eve calls, total eve charge, total night calls, total night charge, total intl calls, total intl charge, number customer service calls.

Formulae used for normalization is

$$Value_{new} = \frac{Value - minValue}{maxValue - minValue}$$

Now our data is ready for applying models. We will use several machine learning classifications models that reads the customer data and predicts who will churn who will not churn with highest accuracy.

2.2 Modeling

2.2.1 Model Selection

After a thorough pre processing we will be using some classifications models on our processed data to predict the target variable.

Decision Tree: Decision tree is a rule. Each branch connects nodes with “and” and multiple branches are connected by “or”. It can be used for classification and regression. It is a Supervised machine learning algorithm. Accept continuous and categorical variables as independent variables. Extremely easy to understand by the business users. There are many types of decision trees. We are using C5.0 model which is entropy based. When we applied this model on our train data, we got certain rules which is provided in a text file. You can also visualize the decision tree with the dot file provided in python code. The accuracy of the model can be seen as following.

```
C50_Predictions
       1    2
  1 1331  112
  2   73  151

               Accuracy : 0.889
                 95% CI : (0.873, 0.9037)
    No Information Rate : 0.8422
    P-Value [Acc > NIR] : 2.662e-08

                  Kappa : 0.5556
```

```
 Mcnemar's Test P-Value : 0.005209

            Sensitivity : 0.9480
            Specificity : 0.5741
         Pos Pred Value : 0.9224
         Neg Pred Value : 0.6741
             Prevalence : 0.8422
         Detection Rate : 0.7984
   Detection Prevalence : 0.8656
      Balanced Accuracy : 0.7611

       'Positive' Class : 1
```

Random Forest: Random Forest or decision tree forests is an ensemble learning method for classification, regression and other tasks. It consists of an arbitrary number of simple trees, which are used to determine the final outcome. For classification problems, the ensemble of simple trees vote for the most popular class. Using tree ensembles can lead to significant improvement in prediction accuracy (i.e., better ability to predict new data cases). The goal of using a large number of trees is to train enough that each feature has a chance to appear in several models. We can see certain rules of random forest in the R code. The accuracy of the model is as follows.

```
RF_Predictions
       1    2
  1 1434    9
  2  158   66

               Accuracy : 0.9058
                 95% CI : (0.8844, 0.9138)
    No Information Rate : 0.955
    P-Value [Acc > NIR] : 1

                  Kappa : 0.4011
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9008
            Specificity : 0.8800
         Pos Pred Value : 0.9938
         Neg Pred Value : 0.2946
             Prevalence : 0.9550
         Detection Rate : 0.8602
   Detection Prevalence : 0.8656
      Balanced Accuracy : 0.8904

       'Positive' Class : 1
```

Logistic Regression: Logistic regression is the appropriate regression analysis to conduct when the dependent variable is binary. Cases where the dependent variable has more than two outcome categories may be analyzed in multinomial logistic regression, or, if the multiple categories are ordered, in ordinal logistic regression. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. The summary of logistic model can be seen as following:

```
Call:
glm(formula = Churn ~ ., family = "binomial", data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.9135  -0.4977  -0.3120  -0.1659   3.0484
```

```
Coefficients:
                         Estimate Std. Error z value Pr(>|z|)
(Intercept)              -9.64158    0.94641 -10.188  < 2e-16 ***
state2                    0.33591    0.76417   0.440 0.660242
state3                    0.90858    0.75353   1.206 0.227907
state4                    0.12055    0.84475   0.143 0.886523
state5                    1.82774    0.78219   2.337 0.019455 *
state6                    0.67607    0.76260   0.887 0.375329
state7                    1.02180    0.72631   1.407 0.159477
state8                    0.69517    0.80924   0.859 0.390319
state9                    0.76200    0.74959   1.017 0.309366
state10                   0.59574    0.76159   0.782 0.434081
state11                   0.67805    0.77818   0.871 0.383574
state12                  -0.21226    0.89560  -0.237 0.812657
state13                   0.23602    0.90325   0.261 0.793857
state14                   0.87469    0.74790   1.170 0.242197
state15                  -0.20620    0.83322  -0.247 0.804542
state16                   0.44382    0.75382   0.589 0.556026
state17                   1.07190    0.73040   1.468 0.142229
state18                   0.80540    0.76596   1.051 0.293036
state19                   0.56522    0.83594   0.676 0.498945
state20                   1.17571    0.74362   1.581 0.113865
state21                   1.14453    0.71710   1.596 0.110481
state22                   1.35342    0.72832   1.858 0.063131 .
state23                   1.38801    0.71413   1.944 0.051940 .
state24                   1.17067    0.71585   1.635 0.101974
state25                   0.59902    0.77472   0.773 0.439402
state26                   1.36003    0.72798   1.868 0.061731 .
state27                   1.87028    0.71735   2.607 0.009128 **
state28                   0.60716    0.75459   0.805 0.421041
state29                   0.15582    0.79713   0.195 0.845017
state30                   0.32490    0.80534   0.403 0.686631
state31                   1.19175    0.76847   1.551 0.120951
state32                   1.59468    0.70979   2.247 0.024660 *
state33                   0.47528    0.78759   0.603 0.546203
state34                   1.25400    0.72542   1.729 0.083869 .
state35                   1.16716    0.72037   1.620 0.105184
state36                   0.68686    0.74724   0.919 0.357996
state37                   0.88256    0.75423   1.170 0.241942
state38                   0.78009    0.73631   1.059 0.289392
state39                   1.15983    0.77995   1.487 0.137001
state40                  -0.10247    0.81983  -0.125 0.900530
state41                   1.77941    0.73736   2.413 0.015813 *
state42                   0.83526    0.76194   1.096 0.272981
state43                   0.28253    0.82136   0.344 0.730858
state44                   1.65240    0.70834   2.333 0.019659 *
state45                   1.05006    0.74417   1.411 0.158228
state46                  -0.43502    0.82288  -0.529 0.597044
state47                   0.10104    0.77844   0.130 0.896728
state48                   1.42380    0.72465   1.965 0.049437 *
state49                   0.28028    0.78093   0.359 0.719666
state50                   0.58562    0.73346   0.798 0.424620
state51                   0.30294    0.75489   0.401 0.688202
account.length            0.23625    0.34713   0.681 0.496135
area.code                -0.06218    0.13775  -0.451 0.651682
international.plan2       2.18813    0.15328  14.275  < 2e-16 ***
voice.mail.plan2         -2.10715    0.59311  -3.553 0.000381 ***
number.vmail.messages     1.91107    0.94922   2.013 0.044082 *
total.day.calls           0.66721    0.47156   1.415 0.157102
total.day.charge          4.59878    0.38909  11.819  < 2e-16 ***
total.eve.calls           0.16764    0.49097   0.341 0.732772
```

```
total.eve.charge               2.82487    0.43058   6.561 5.36e-11 ***
total.night.calls              0.02657    0.41536   0.064 0.948990
total.night.charge             1.46040    0.42800   3.412 0.000645 ***
total.intl.calls              -1.79972    0.51370  -3.503 0.000459 ***
total.intl.charge              1.67077    0.42178   3.961 7.46e-05 ***
number.customer.service.calls  4.83059    0.36840  13.112  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2758.3  on 3332  degrees of freedom
Residual deviance: 2072.0  on 3268  degrees of freedom
AIC: 2202

Number of Fisher Scoring iterations: 6

logit_Predictions
       0    1
  1 1374   69
  2  158   66
```

This model works with an accuracy of **86.38%**

K-NN Implementation: K-Nearest Neighbors algorithm (KNN) is a non-parametric method used for classification and regression. KNN is a type of instance-based learning or lazy learning and simplest of all machine learning algorithms. Even with such simplicity, it can give highly competitive results. It is more widely used in classification problems in the industry. KNN fairs across all parameters of considerations. It is commonly used for its easy of interpretation and low calculation time. K-NN is a lazy learner because it doesn’t learn a discriminative function from the training data but “memorizes” the training dataset instead. The accuracy of KNN model is

```
KNN_Predictions
       1    2
   1 1430  210
   2   13   14

               Accuracy : 0.8662
                 95% CI : (0.8489, 0.8822)
    No Information Rate : 0.8656
    P-Value [Acc > NIR] : 0.4892

                  Kappa : 0.0851
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9910
            Specificity : 0.0625
         Pos Pred Value : 0.8720
         Neg Pred Value : 0.5185
             Prevalence : 0.8656
         Detection Rate : 0.8578
   Detection Prevalence : 0.9838
      Balanced Accuracy : 0.5267

       'Positive' Class : 1
```

Naïve Bayes: It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. The accuracy of Naïve Bayes model is

```
        predicted
observed    1    2
       1 1349   94
       2  120  104

               Accuracy : 0.8716
                 95% CI : (0.8546, 0.8873)
    No Information Rate : 0.8812
    P-Value [Acc > NIR] : 0.89327

                  Kappa : 0.4197
 Mcnemar's Test P-Value : 0.08746

            Sensitivity : 0.9183
            Specificity : 0.5253
         Pos Pred Value : 0.9349
         Neg Pred Value : 0.4643
             Prevalence : 0.8812
         Detection Rate : 0.8092
   Detection Prevalence : 0.8656
      Balanced Accuracy : 0.7218

       'Positive' Class : 1
```

Therefore from all the above models we can see that random forest classifier gives us the highest accurac y in predicting the target variable. Now we must see several error metrics and fix our model on the test dat a.

Chapter 3

Conclusion

3.1 Model Evaluation

Model evaluation is done on basis of evaluation metrics or error metrics. Evaluation metrics explain the performance of a model. An important aspect of evaluation metrics is their capability to discriminate among model results. Simply, building a predictive model is not our motive. But, creating and selecting a model which gives high accuracy on out of sample data. Hence, it is crucial to check accuracy or other metric of the model prior to computing predicted values. In our data as we applied classification models we have error metrics like confusion matrix out of which we can check specificity, recall, false negative rate, false positive rate. We will evaluate error metrics for each of the model that is applied

Confusion matrix has True Positive(TP), True Negative(TN), False Positive(FP), False Negative(FN)

True Positive is the number of correct predictions that an instance is Yes,
False Negative is the number of incorrect predictions that an instance is No,
False Positive is the number of incorrect of predictions that an instance Yes, and
True Negative is the number of correct predictions that an instance is No.

Decision Tree:

```
Confusion Matrix
                        C50_Predictions

                             1    2
                        1 1331  112
                        2   73  151
```

Accuracy: (TP+TN)/Total observations = 0.889
Sensitivity or Recall: TP/TP+FN = 0.9480
Specificity: TN/TN+FP = 0.5741
False positive rate: FP/FP+TN = 0.6741
False Negative rate: FN/FN+TP = 0.3259

Random Forest:

```
Confusion Matrix
                        RF_Predictions

                             1    2
                        1 1434    9
                        2  148   76
```

Accuracy: (TP+TN)/Total observations = 0.9058
Sensitivity or Recall: TP/TP+FN = 0.9064
Specificity: TN/TN+FP = 0.8941
False positive rate: FP/FP+TN = 0.3393
False Negative rate: FN/FN+TP = 0.6607

Logistic Regression:

```
Confusion Matrix
                        logit_Predictions

                             1    2
                        1 1374   69
                        2  158   66
```

Accuracy: (TP+TN)/Total observations = 0.8638
Sensitivity or Recall: TP/TP+FN = 0.9064
Specificity: TN/TN+FP = 0.8941
False positive rate: FP/FP+TN = 0.2947
False Negative rate: FN/FN+TP = 0.7053

KNN Implementation:

```
Confusion Matrix

                        KNN_Predictions
                             1    2
                        1 1430  210
                        2   13   14
```

Accuracy: (TP+TN)/Total observations = 0.8662
Sensitivity or Recall: TP/TP+FN = 0.9910
Specificity: TN/TN+FP = 0.0625
False positive rate: FP/FP+TN = 0.5185
False Negative rate: FN/FN+TP = 0.4814

Naïve Bayes:

```
Confusion Matrix

                        KNN_Predictions

                             1    2
                        1 1349   94
                        2  120  104
```

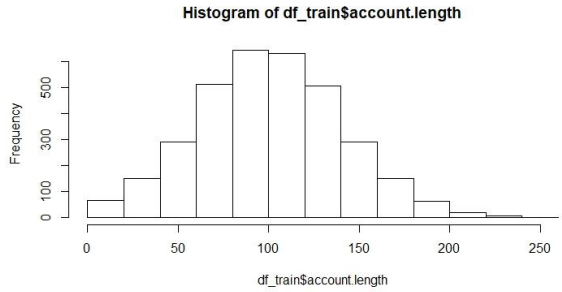
Accuracy: (TP+TN)/Total observations = 0.8716
Sensitivity or Recall: TP/TP+FN = 0.9183
Specificity: TN/TN+FP = 0.5253
False positive rate: FP/FP+TN = 0.4643
False Negative rate: FN/FN+TP = 0.5357

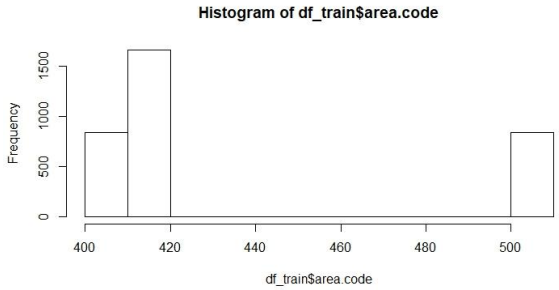
3.2 Model Selection

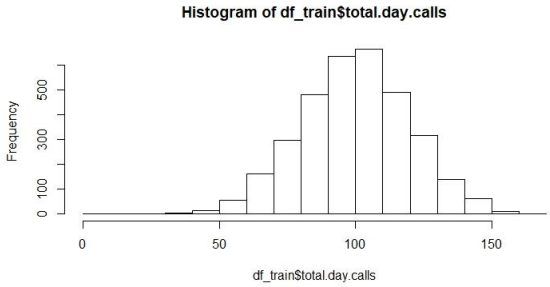
We can see that all models perform comparatively on average and therefore we select either decision tree or random forest classifier models for better prediction.

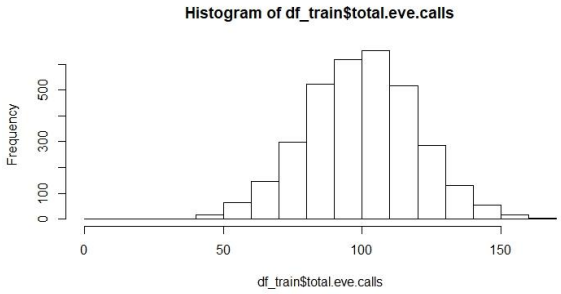
Appendix A - Extra Figures

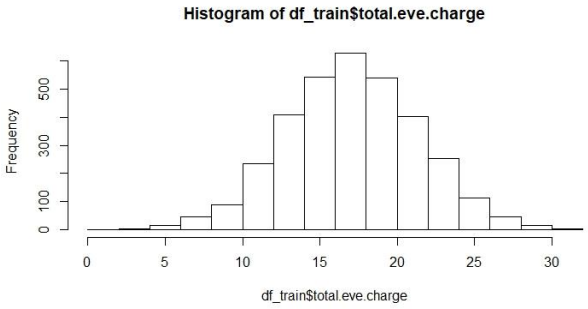
Normality check plots of various numerical variables:

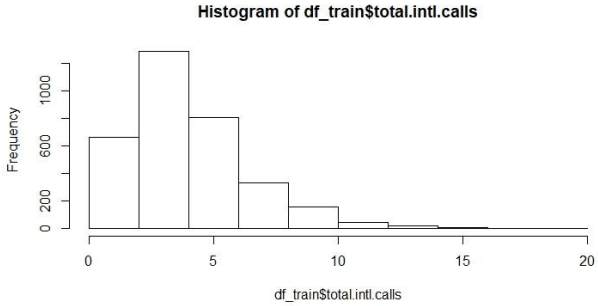


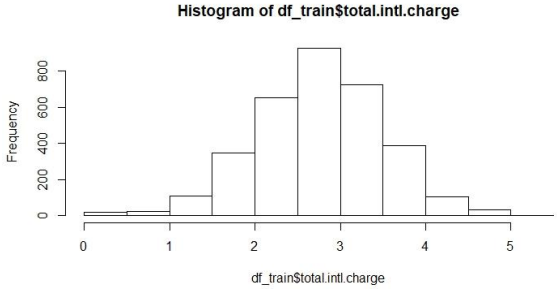


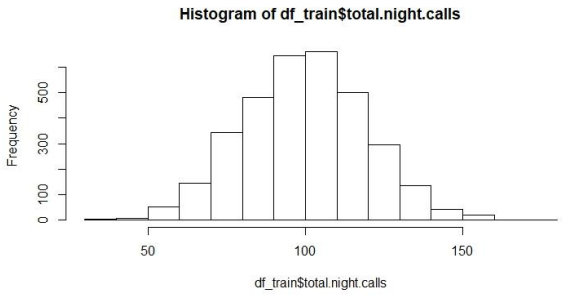


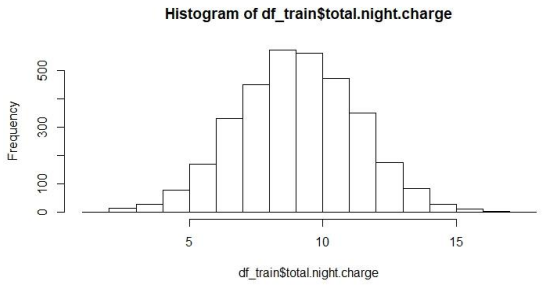


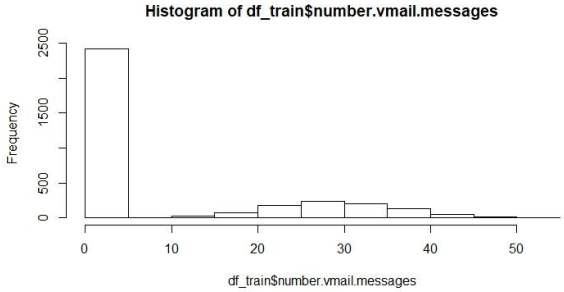


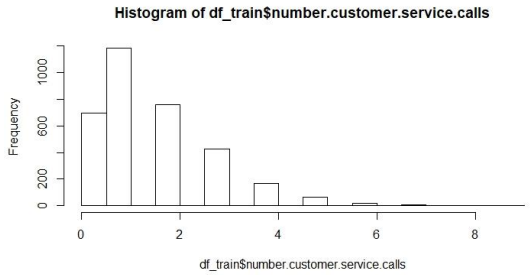












Appendix B - Code

R code

```
rm(list = ls())
setwd("C:/Users/chinna/Desktop/Data Science/Project")

## Read the data
df_train = read.csv("Train_data.csv", header = T)
df_test = read.csv("Test_data.csv", header = T)

#structure od data
str(df_train)

#Load Libraries
x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50", "dummies",
"e1071", "Information",
    "MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine', 'inTrees')

lapply(x, require, character.only = TRUE)
rm(x)


#Missing value analysis
missing_train = data.frame(apply(df_train,2,function(x){sum(is.na(x))}))
missing_test = data.frame(apply(df_test,2,function(x){sum(is.na(x))}))

##Data Manupulation; convert string categories into factor numeric
#assigning levels to categorical variables
fnames = c("state","international.plan","voice.mail.plan","Churn")

for(i in fnames){
  if(class(df_train[,i])== 'factor')
```

```
{
   df_train[,i] = factor(df_train[,i], labels = (1:length(levels(factor(df_train[,i]))))
  }
}

for(i in fnames){
  if(class(df_test[,i])== 'factor'){
    df_test[,i] = factor(df_test[,i], labels = (1:length(levels(factor(df_test[,i])))))
  }
}
rm(fnames)
rm(i)


numeric_index = sapply(df_train,is.numeric) #selecting only numeric
numeric_data = df_train[,numeric_index]
cnames = colnames(numeric_data)

## Correlation Plot
corrgram(df_train[,numeric_index], order = F,
         upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")

## Chi-squared Test of Independence
factor_index = sapply(df_train,is.factor)
factor_data = df_train[,factor_index]

for (i in 1:4)
{
  print(names(factor_data)[i])
  print(chisq.test(table(factor_data$Churn,factor_data[,i])))
}

## Dimension Reduction
df_train = subset(df_train, select = -c(total.day.minutes, total.eve.minutes, total.night.minutes,
total.intl.minutes, phone.number))
df_test = subset(df_test, select = -c(total.day.minutes, total.eve.minutes, total.night.minutes,
total.intl.minutes, phone.number))

#normality check
hist(df_train$number.customer.service.calls)


#Normalisation
cnames                                                                                                    =
c("account.length","area.code","number.vmail.messages","total.day.calls","total.day.charge",
          "total.eve.calls","total.eve.charge","total.night.calls","total.night.charge","total.intl.calls",
"total.intl.charge",
          "number.customer.service.calls")

for(i in cnames){
  print(i)
  df_train[,i] = (df_train[,i] - min(df_train[,i]))/
    (max(df_train[,i] - min(df_train[,i])))
}

for(i in cnames){
  print(i)
  df_test[,i] = (df_test[,i] - min(df_test[,i]))/
    (max(df_test[,i] - min(df_test[,i])))
}
```

```
rmExcept(c("df_train","df_test"))

train = df_train
test = df_test

##Decision tree for classification
#Develop Model on training data
C50_model = C5.0(Churn ~., train, trials = 50, rules = TRUE)

#Summary of DT model
summary(C50_model)

#write rules into disk
write(capture.output(summary(C50_model)), "c50Rules.txt")

#Lets predict for test cases
C50_Predictions = predict(C50_model, test[,-16], type = "class")

##Evaluate the performance of classification model
ConfMatrix_C50 = table(test$Churn, C50_Predictions)
confusionMatrix(ConfMatrix_C50)

#False Negative rate
FNR = FN/FN+TP

#Accuracy: 88.9%
#FNR: 32.58%

###Random Forest
RF_model = randomForest(Churn ~ ., train, importance = TRUE, ntree = 150)

#Extract rules fromn random forest
#transform rf object to an inTrees' format
treeList = RF2List(RF_model)
#
#Extract rules
exec = extractRules(treeList, train[,-16])  # R-executable conditions

#Visualize some rules
exec[1:2,]

#Make rules more readable:
readableRules = presentRules(exec, colnames(train))
readableRules[1:2,]

#Get rule metrics
ruleMetric = getRuleMetric(exec, train[,-16], train$Churn)  # get rule metrics

#evaulate few rules
ruleMetric[1:2,]

#Presdict test data using random forest model
RF_Predictions = predict(RF_model, test[,-16])

##Evaluate the performance of classification model
ConfMatrix_RF = table(test$Churn, RF_Predictions)
confusionMatrix(ConfMatrix_RF)
```

```
#False Negative rate
FNR = FN/FN+TP

#Accuracy = 90.58
#FNR = 66.96

#Logistic Regression
logit_model = glm(Churn ~ ., data = train, family = "binomial")

#summary of the model
summary(logit_model)

#predict using logistic regression
logit_Predictions = predict(logit_model, newdata = test, type = "response")

#convert prob
logit_Predictions = ifelse(logit_Predictions > 0.5, 1, 0)


##Evaluate the performance of classification model
ConfMatrix_LR = table(test$Churn, logit_Predictions)

#False Negative rate
FNR = FN/FN+TP

#Accuracy: 86.38
#FNR: 70.53

##KNN Implementation
library(class)

#Predict test data
KNN_Predictions = knn(train[, 1:15], test[, 1:15], train$Churn, k = 7)

#Confusion matrix
ConfMatrix_Knn = table(KNN_Predictions, test$Churn)
confusionMatrix(ConfMatrix_Knn)

#False Negative rate
FNR = FN/FN+TP

#Accuracy = 86.62
#FNR = 48.14

#naive Bayes
library(e1071)

#Develop model
NB_model = naiveBayes(Churn ~ ., data = train)

#predict on test cases #raw
NB_Predictions = predict(NB_model, test[,1:15], type = 'class')

#Look at confusion matrix
Confmatrix_NB = table(observed = test[,16], predicted = NB_Predictions)
confusionMatrix(Confmatrix_NB)

#Accuracy: 87.16
#FNR: 53.57
```

Python Code

```
# coding: utf-8

# In[ ]:
#Load libraries
import os
import pandas as pd
import numpy as np
from fancyimpute import KNN
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency
import seaborn as sns
from random import randrange, uniform

# In[ ]:
#Setting path
os.chdir("C:/Users/chinna/Desktop/Data Science/Project")

# In[ ]:
#loading train and test data
df_train = pd.read_csv("Train_data.csv")
df_test = pd.read_csv("Test_data.csv")


# In[ ]:
#missing value analysis
#storing number of missing values of each variable in dataframe
missing_train = pd.DataFrame(df_train.isnull().sum())
missing_test = pd.DataFrame(df_test.isnull().sum())


# In[ ]:
missing_train


# In[ ]:
#distribution of target variable
y = df_train["Churn"].value_counts()
sns.barplot(y.index, y.values)


# In[ ]:
#target variable w r t categorical variable
#state vs churn
df_train.groupby(["state", "Churn"]).size().unstack().plot(kind='bar', stacked=True, figsize=(30,10))


# In[ ]:
#areacode vs churn
df_train.groupby(["area code", "Churn"]).size().unstack().plot(kind='bar', stacked=True, figsize=(5,5))


# In[ ]:
#international plan vs churn
df_train.groupby(["international plan", "Churn"]).size().unstack().plot(kind='bar', stacked=True,
figsize=(5,5))
```

```
# In[ ]:
#voice mail plan vs churn
df_train.groupby(["voice mail plan", "Churn"]).size().unstack().plot(kind='bar', stacked=True, figsize=(5,5))


# In[ ]:
#assigning levels to categorical varibales
for i in range(0, df_train.shape[1]):
    if(df_train.iloc[:,i].dtypes == 'object'):
        df_train.iloc[:,i] = pd.Categorical(df_train.iloc[:,i])
        df_train.iloc[:,i] = df_train.iloc[:,i].cat.codes

for i in range(0, df_test.shape[1]):
    if(df_test.iloc[:,i].dtypes == 'object'):
        df_test.iloc[:,i] = pd.Categorical(df_test.iloc[:,i])
        df_test.iloc[:,i] = df_test.iloc[:,i].cat.codes


# In[ ]:
df_test.head(10)


# In[ ]:
#storing target variable
train_targets = df_train.Churn
test_targets = df_test.Churn


# In[ ]:
#combining train and test data for data prepocessing
combined = df_train.append(df_test)


# In[ ]:
print(combined.shape, df_train.shape, df_test.shape)


# In[ ]:
cnames = ["account length","area code","number vmail messages","total day minutes","total day calls","total day charge",
          "total eve minutes","total eve calls","total eve charge","total night minutes","total night calls",
          "total night charge","total intl minutes","total intl calls", "total intl charge",
          "number customer service calls"]


# In[ ]:
df_corr = combined.loc[:,cnames]


# In[ ]:
#correlation analysis
#set height and width of plot
f , ax = plt.subplots(figsize = (7,5))

#generate correlation matrix
corr = df_corr.corr()
#plot using seaborn library
```

```
sns.heatmap(corr, mask=np.zeros_like(corr,dtype=np.bool),cmap =
sns.diverging_palette(220,10,as_cmap=True),
        square = True, ax=ax)


# In[ ]:
cat_names = ["state","phone number","international plan","voice mail plan","Churn"]


# In[ ]:
#chi square test of independence
for i in cat_names:
    print(i)
    chi2, p, dof, ex = chi2_contingency(pd.crosstab(combined['Churn'],combined[i]))
    print(p)


# In[ ]:
#dropping unnecessary variables
combined = combined.drop(["total day minutes", "total eve minutes", "total night minutes", "total intl
minutes",
                "phone number","Churn"], axis = 1)


# In[ ]:
combined.shape


# In[ ]:
cnames = ["account length","area code","number vmail messages","total day calls","total day charge",
        "total eve calls","total eve charge","total night calls","total night charge","total intl calls",
        "total intl charge", "number customer service calls"]


# In[ ]:
#normalization
for i in cnames:
    print(i)
    combined[i] = (combined[i]-min(combined[i]))/(max(combined[i])-min(combined[i]))


# In[ ]:
combined.head(10)


# In[ ]:
combined.shape


# In[ ]:
#loading libraries for model
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import train_test_split


# In[ ]:
#splitting combined data to train and test
train = combined[:3333]
```

```
test = combined[3333:]


# In[ ]:
#decision tree model
c50_model = tree.DecisionTreeClassifier(criterion = 'entropy').fit(train, train_targets)

c50_pred = c50_model.predict(test)


# In[ ]:
c50_pred


# In[ ]:
#dot file to look at decision tree
dotfile = open("pt.dot", 'w')
df = tree.export_graphviz(c50_model, out_file=dotfile, feature_names = train.columns)


# In[ ]:
#testing accuracy of model
from sklearn.metrics import confusion_matrix

CM = pd.crosstab(test_targets, c50_pred)
CM


# In[ ]:
TN = CM.iloc[0,0]
FN = CM.iloc[1,0]
TP = CM.iloc[1,1]
FP = CM.iloc[0,1]

accuracy_score(test_targets,c50_pred)*100

#accuracy = 92.32153

#(FN*100)/(FN+TP)

#FNR = 32.142857

#(TP*100)/(TP+FN)

#Recall = 67.8571428


# In[ ]:
#random forest model

from sklearn.ensemble import RandomForestClassifier

RF_model = RandomForestClassifier(n_estimators = 100).fit(train,train_targets)

RF_prediction = RF_model.predict(test)
```

```
# In[ ]:
RF_prediction


# In[ ]:
CM = pd.crosstab(test_targets, RF_prediction)
CM


# In[ ]:
#let us save TP, TN, FP, FN
TN = CM.iloc[0,0]
FN = CM.iloc[1,0]
TP = CM.iloc[1,1]
FP = CM.iloc[0,1]

#check accuracy of model
#accuracy_score(y_test, y_pred)*100
((TP+TN)*100)/(TP+TN+FP+FN)
#accuracy = 94.96100

#(FN*100)/(FN+TP)
#FNR = 33.928571


# In[ ]:
#KNN implementation
from sklearn.neighbors import KNeighborsClassifier

KNN_model = KNeighborsClassifier(n_neighbors = 9).fit(train, train_targets)


# In[ ]:
#predict test cases
KNN_Predictions = KNN_model.predict(test)


# In[ ]:
#build confusion matrix
CM = pd.crosstab(test_targets, KNN_Predictions)
CM


# In[ ]:
#let us save TP, TN, FP, FN
TN = CM.iloc[0,0]
FN = CM.iloc[1,0]
TP = CM.iloc[1,1]
FP = CM.iloc[0,1]

#check accuracy of model
#accuracy_score(y_test, y_pred)*100
((TP+TN)*100)/(TP+TN+FP+FN)
#accuracy = 86.622675


#False Negative rate
#(FN*100)/(FN+TP)
#FNR =97.321428
```

```
# In[ ]:
#Naive Bayes
from sklearn.naive_bayes import GaussianNB

#Naive Bayes implementation
NB_model = GaussianNB().fit(train, train_targets)


# In[ ]:
#predict test cases
NB_Predictions = NB_model.predict(test)


# In[ ]:
#Build confusion matrix
CM = pd.crosstab(test_targets, NB_Predictions)
CM


# In[ ]:
#let us save TP, TN, FP, FN
TN = CM.iloc[0,0]
FN = CM.iloc[1,0]
TP = CM.iloc[1,1]
FP = CM.iloc[0,1]

#check accuracy of model
#accuracy_score(Y_test, y_pred)*100
((TP+TN)*100)/(TP+TN+FP+FN)
#Accuracy = 85.842831433

#False Negative rate
#(FN*100)/(FN+TP)
#FNR = 60.2678571


# In[ ]:
#we will be fixing random forest model as it provides best results
#now we will generate example out for out sample input test data with Random forest predictions

move = pd.DataFrame(RF_prediction)

move = move.rename(columns = {0:'move'})


# In[ ]:
test = test.join(move['move'])


# In[ ]:
test.to_csv("example_output.csv", index = False)
```