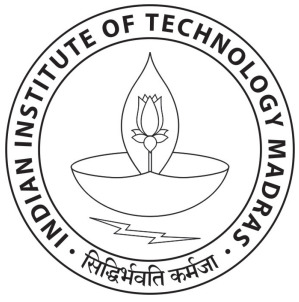

MS6840

DATA WAREHOUSING AND DATA MINING

DEPARTMENT OF MANAGEMENT STUDIES

Costa Rican Household Poverty Level Prediction

Group 1:

Veena Kannan
Kanti Kumari
Kumar Madhav Narayan
Pappu Ram

September 30, 2018

Contents

1 Problem Definition

The major problem in many social programs is to select the right people to give the financial aid.The most poorer sections can't provide necessary details so it is difficult for them to qualify for the same. Costa Rica is a country in Central America with a population of about 4.9 million. One of the most common algorithms used in Latin America is the Proxy Means Test(PMT), in which the observable items in a household such as television, electricity supply, number of educated people etc are used to determine and classify the households into groups (eg: 1-extreme poverty,2-moderate poverty,3-vulnerable households,4 – non vulnerable households).We downloaded the Costa rica household dataset from **Kaggle** for this multiclass classification and built model on the training data and predicted it for the test data.The multiclass classification results can be used by the banks(in this case Inter-American Development Bank) to correctly target the needy people and give the necessary financial aid.

2 Technical Tools

2.1 Programming Languages

Programming language R for data preprocessing and Python(2 and 3 version) to develop classification models have been used in this project.

2.2 Software

We used Spyder, Jupyter Notebook, Sublime Text and R Studio for source code editing. LaTeX has been used for report writing.

2.3 Hardware

Two types of computer systems were used:

1. Intel® Xeon(R) CPU E5-1650, 32GB RAM and 12 GB GPU memory, and
2. Intel core i3 processor, RAM 8GB.

3 Data Description

The given training data has 9557 rows with 143 columns(142 features + 1 proverty level of household) while test set contains 23856 object with 142 features. Poverty level of household is target variable which we will be estimating using other 143 estimators. Three kinds of data types were present in the features of training data (Figure 1).

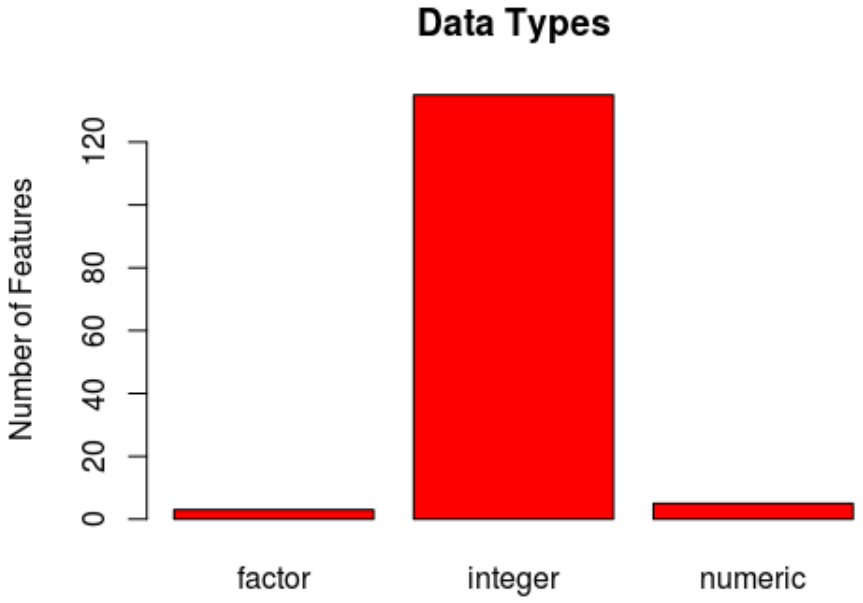

Figure 1: Histogram plot to display the data types.

On the basis of subject of which variables represent we can divide types of given features as:

3.1 Predictors available in dataset

There are 143 features in the given dataset describing different aspects of the households.So we have grouped the features and is presented in the table below .

Table 1: Features present in the dataset for Costa Rican proverty detection

S.No.	Variable category	Variables
1	Household	$r4t3$, $tamhog$, $tamviv$, $hhsize$,$hogar\_total$, $hogar\_mayor$, $idhogar$
2	Devices and Appliance variables	$refrig$, $v18q$, $v18q1$, $computer$,$television$,$mobilephone$,$qmobilephone$
3	Building material	$pareddes$, $paredblolad$, $paredzocalo$, $paredpreb$, $paredmad$, $paredzinc$, $paredfibras$, $paredother$, $pisomoscer$, $pisocemento$, $pisoother$, $pisonatur$, $pisonotiene$, $pisomadera$, $techozinc$, $techoentrepiso$, $techocane$, $techootro$
4	water, electricity, waste disposal and sewerage related variables	$abastaguadentro$, $abastaguafuera$, $abastaguano$, $public$, $planpri$, $noelec$, $coopele$, $sanitario1$, $sanitario2$, $sanitario3$, $sanitario5$, $sanitario6$, $energcocinar1$, energcocinar2, energcocinar3, $energcocinar4$, $elimbasu1$, $elimbasu2$, $elimbasu3$, elimbasu4, elimbasu5, $elimbasu6$
5	Education of members	instlevel1, instlevel2,instlevel3, instlevel4, instlevel5, instlevel6, instlevel7, instlevel8, instlevel9, edjefa, edjefe, meaneduc, escolari, rez_esc
6	Family member variables	dis, male, female, estadocivil1, estadocivil2, estadocivil3, estadocivil4, estadocivil5, estadocivil6, estadocivil7, parentesco1, parentesco2, parentesco3, parentesco4, parentesco5, parentesco6, parentesco7, parentesco8, parentesco9, parentesco10, parentesco11, parentesco12
7	Demographic variables	lugar1, lugar2, lugar3, lugar4, lugar5, lugar6, area1, area2, age, SQBescolari, SQBage, SQBhogar_total, SQBedjefe, SQBhogar_nin, SQBovercrowding, SQBdependency, SQBmeaned, agesq, r4h1, r4h2, r4h3, r4m1, r4m2, r4m3, r4t1, r4t2, r4t3, hogar_nin,hogar_adul, dependency, overcrowding, tipovivi1, tipovivi2, tipovivi3, tipovivi4, tipovivi5
8	House condition variables	cielorazo, epared1, epared2, epared3, etecho1, etecho2, etecho3, eviv1, eviv2, eviv3
7	House interior varaibles	bedrooms, rooms, hacdor, v14a, hacapo

3.2 Data Visualization

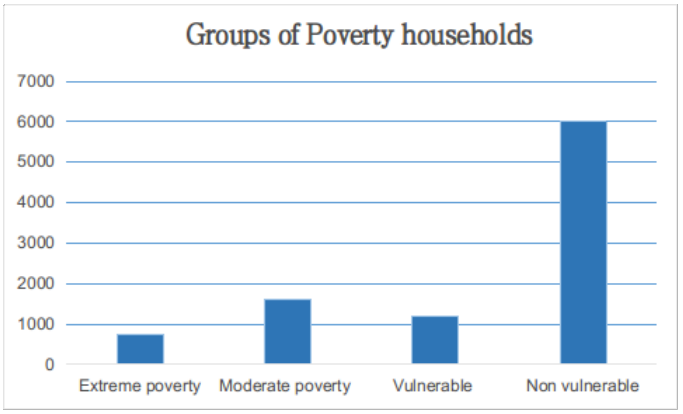

Figure 2: Histogram plot to display different poverty levels.

Given dataset is not balanced in terms of poverty level of individuals. The count of "non vulnerable" is very high as compared to the other three levels (Figure 2).

Similarly, the plot in Figure 3 tells many of the households does not own any tablet. Figure 4 conveys that the count of households who owns and have fully paid house are present in very high percent among the total individuals.

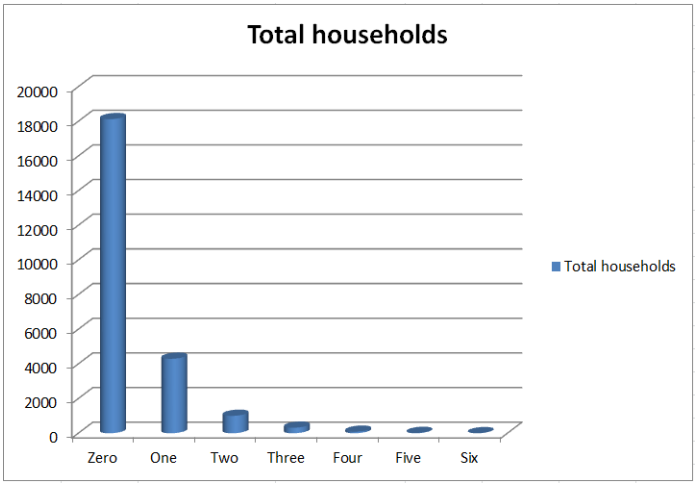

Figure 3: Histogram plot for total number of households based on the count of tablets they are having.

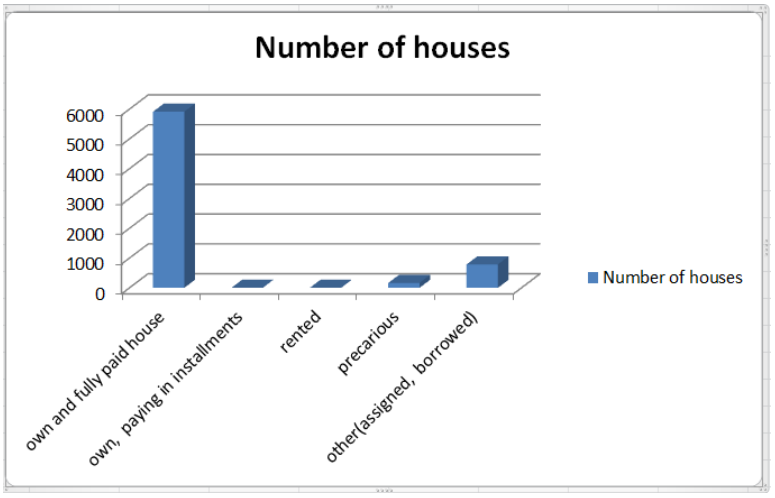

Figure 4: Histogram plot representing total number of household for different modes of rent payment.

4 Data Preprocessing

4.1 Data cleaning

Variables $SQBescolari$, $SQBage$, $SQBhogar\_total$,$SQBedjefe$, $SQBhogar\_nin$, $SQBovercrowding$, $SQBdependency$ and $SQBmeaned$ are square of the variable $escolari$, age , $hogar\_total$, $edjefe$, $hogar\_nin$, $overcrowding$, $dependency$ and $meaneduc$ respectively. Hence all these squared variables has been removed. Also, both columns $agesq$ and $SQBage$ represents squared age hence $agesq$ was also removed from the dataset.

Since first attribute represent unique identifier for each household (for each object in the given dataset), considering this attribute will not be considered in the training of model we did not consider it during model development process hence this variable was deleted. While house ID is common for members of the same family, we converted it as numerical variable and considered for model building.

The columns $edjefe$ and $edjefa$ has yes and no values were replaced with their given corresponding numerical values.

Dependency attribute is the ratio of children below 19 as well as adults above 65

to the adults aged between 19 to 65 years. We recalculated the dependency using the given formula and few features of the same dataset and found out that there were households with no adults aged between 19 to 65 years so dependency of these households were very high hence it was kept as 8. Attribute value 'yes' shows the equal number of dependents and non-dependents while 'no' represent there is not any dependent in the household. Hence, 'yes' and 'no' were replaced with 1 and 0 (Figure 5).

Corrected Dependency	Dependency
0	no
#DIV/0!	8
#DIV/0!	8
1	yes
1	yes
.	.
.	.
.	.
1	yes

Figure 5: Content of column Dependency with with the evaluated dependency.

We have find that there is discrepancy between the target value of the head of the household and other members in 41 cases. So the Target attribute values of the others members were replaced with that of the head of the same household as informed by the data owners.

The attribute *elimbasu*5 (rubbish disposal mainly by throwing in river by household) has only one value 0, therefore it will not be having any effect on model training. Hence we removed this column from the dataset.

After data cleaning process we have reduced total number of features from 143 to 133.

4.2 Missing Data Imputation

We have find that the dataset has five variable with missing data these variable are: *rez_esc*, *v*18*q*1 , *v*2*a*1, *meaneduc* and *SQBmeaned* with 7928, 7342, 6860, 5 and 5 missing values respectively as shown in Figure 6.

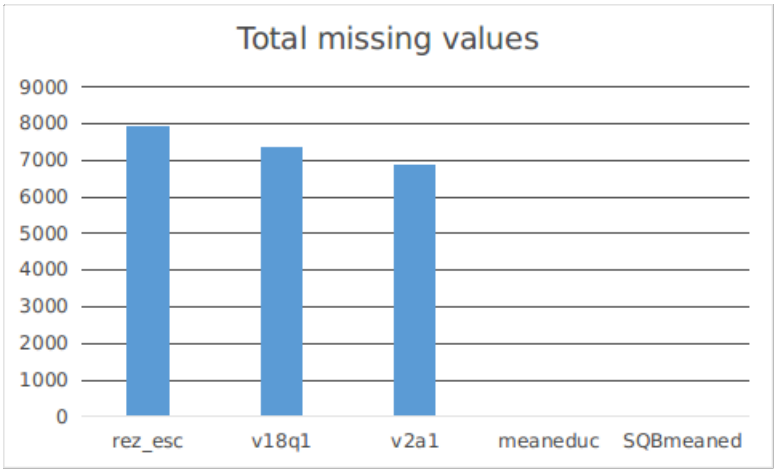

Figure 6: Histogram using variables present in the dataset with the total number of rows with missing values present for these features.

The imputation is done as described below:

1. The attribute *v*18*q* is a binary variable representing whether household owns a tablet or not, while feature *v*18*q*1 having the total number of tablets owned. We have observed that *v*18*q*1 has NA when household does not own tablet hence we decided to replace NA for this attribute with 0.

2. The feature *v*2*a*1 represents Monthly rent payment. From the dataset we observed that most of the missing values in this variable are present when households own a house. So people who own the house we replaced value of feature *v*2*a*1 with zero. With this the total NA values in the mentioned column reduced from 6860 to 949 which we replaced with the feature median value.

3. *Rez_esc* represents the years behind the school. Indiduals with age less than 7 years and greater than 19 years will not be in school. Hence the value for variable *Rez_esc* for individuals of this age group was replaced with zero. With this we reduced missing cell count for it to 350 .

4. The column *mean_educ* contains the average years of education of adults in the household. Since this variable is continuous we imputed it with its column mean.

5 Feature Engineering

5.1 Removal of redundant features

In order to remove the redundant household variables we found out the correlation among them variables.

The dataset consists of different household variables which are:

1. *r4t3* :Total persons in the household
2. *tamhog* : size of the household
3. *tamviv* : number of persons living in the household
4. *hhsize* : household size
5. *hogar_total* : no of total individuals in the household

	r4t3	tamhog	tamviv	hhsize	hogar_total
r4t3	1				
tamhog	0.999368	1			
tamviv	0.981736	0.981172	1		
hhsize	0.999368	1	0.981172	1	
hogar_total	0.999368	1	0.981172	1	1

Figure 7: Correlation matrix for household variables present in the datasset.

r4t3 ,*tamhog* ,*hogar_total* are highly correlated and hence removed.The only variables need to be retained are *tamviv* and *hhsize* as the number of households living in the house need not be the same as the *hhsize*.

After this process, total number of features in dataset have been reduced to 130.

5.2 Aggregation of categorical variables into ordinal

Many variables in the given dataset can be reduced into ordinal variables.

5.2.1 Variables indicating electricity connection

The features *noelec*, *coopele*, *public* and *planpri* are binary variables representing electric connection to the house which can be put in an order from no electricity to electric connection from private plant in one variable as:
0: No electricity
1: Electricity from cooperative
2: Electricity from CNFL, ICA, ESPH/JASEC
3: Electricity from private plant

After aggregation the new variable was left with 15 missing values which was imputed with the variable mode. Total number of variables after reduction was 127. R code for same is as shown in Figure 8.

Similar code have been used to aggregate other potential ordinal variables into one.

5.2.2 Quality of wall,roof and floor

0: Bad
1:Regular
2:Good
Total number of variables after reduction :121

5.2.3 Education ordinal variable

instlevel_ indicating the level of education from 1(indicating no education)to 9(education upto postgraduate level.So an ordinal variable was created thus dropping the rest of variables.
Total number of variable after reduction : 113

```
dataset$elec = NA
for(i in 1:nrow(dataset)) {
  if (dataset[i,'noelec'])
  {
    dataset[i,'elec'] = 0
  }
  else if(dataset[i, 'coopele'])
  {
    dataset[i,'elec'] = 1
  }
  else if(dataset[i, 'public'])
  {
    dataset[i,'elec'] = 2
  }
  else if(dataset[i, 'planpri'])
  {
    dataset[i,'elec'] = 3
  }
}
dataset$noelec = NULL
dataset$coopele = NULL
dataset$public = NULL
dataset$planpri = NULL
dataset[is.na(dataset$elec), 'elec'] = Mode(dataset$elec)
```

Figure 8: R code to aggregate different levels present as different feature for electric connection, into one column.

5.2.4 Gender

There were two different variables indicating female and male. So one variable(male) was dropped.
Total number of variables after reduction is 112.

5.2.5 Area of households

Two variables area1 and area2 representing urban and rural area of household. Both variables are mutually exclusive hence combined to a single categorical variable area with urban as 1 and rural region as 0.
Total number of variables after reduction 111.

5.3 Feature Selection

Important features were identified using 'Boruta' package which uses random forest to estimate the importance of variables present in a given dataset.

5.3.1 Boruta Algorithm

Boruta is a feature selection wrapper algorithm which can work with any classification method and it outputs the variable importance measure . We used Random forest as an algorithm to identify relevant features present in our dataset. After every iteration the algorithm checks whether a particular feature has a higher importance than its shadow features by comparing the Z score and rejects the feature with lowest Z score.We have done Boruta algorithm for 100 iterations.(Kursa, M. B., Rudnicki, W. R. (2010). Feature selection with the Boruta package. J Stat Softw, 36(11), 1-13.)

```
boruta_output = Boruta(formula, data=na.omit(dataset), doTrace=2, maxRuns = 100)
boruta_signif = names(boruta_output$finalDecision[boruta_output$finalDecision %in%
                                                  c("Confirmed", "Tentative")])

print(boruta_signif)
plot(boruta_output, cex.axis=.7, las=2, xlab="", main="Variable Importance")
```

Figure 9: R code for Boruta to get importance of variables present in the preprocessed dataset.

```
 32. run of importance source...
Computing permutation importance.. Progress: 51%. Estimated remaining time: 29 seconds.
Computing permutation importance.. Progress: 98%. Estimated remaining time: 1 seconds.
After 32 iterations, +42 mins:
 confirmed 1 attribute: energcocinar1;
 still have 12 attributes left.
```

Figure 10: Screenshot taken during the variable estimation process.

With the above mentiones process, we identified 95 important predictor features and removed other 15 lowest important variables.

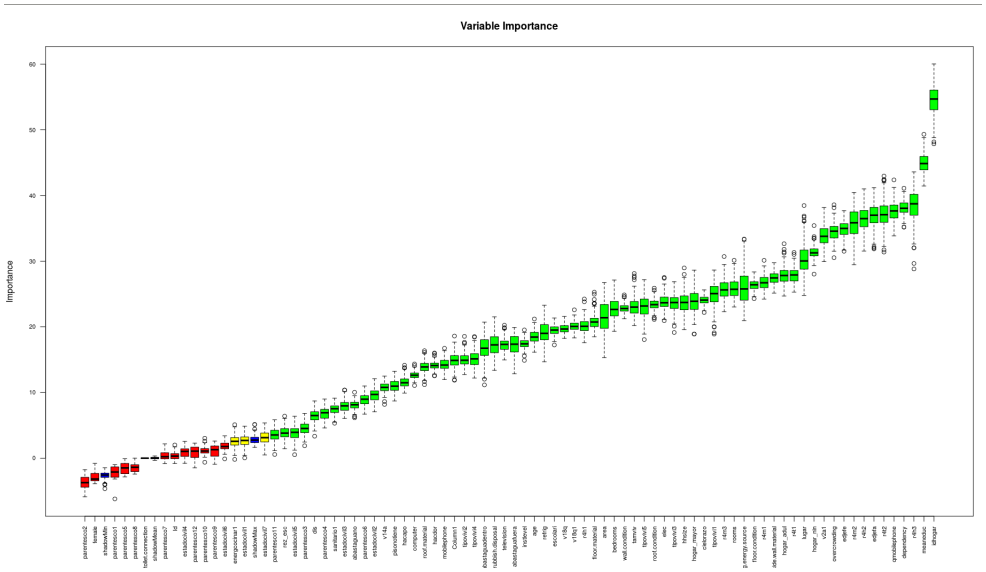

Figure 11: Variable importance plot generated using Boruta for all variables present in the dataset.

6 Data visualization after preprocessing

The four most important features obtained from feature engineering were plotted against the target variables against different levels of poverty. These variables are *dependency*, *Education level*, *mobile phone usage*, and *House rent payment* (Figure 12-15). With these plot we can say that households in extreme poverty level(level 1) has high dependency,low level of education .mobile phone usage and rent payment.

With Figure 12 it is clear that people with very low level of education are extremely poor as well as they are the ones who does not use mobile phone much(Figure 13). Similarly, Figure 14 show extremely poor people are highly dependent while non-vulnerable are independents and pay high rent.

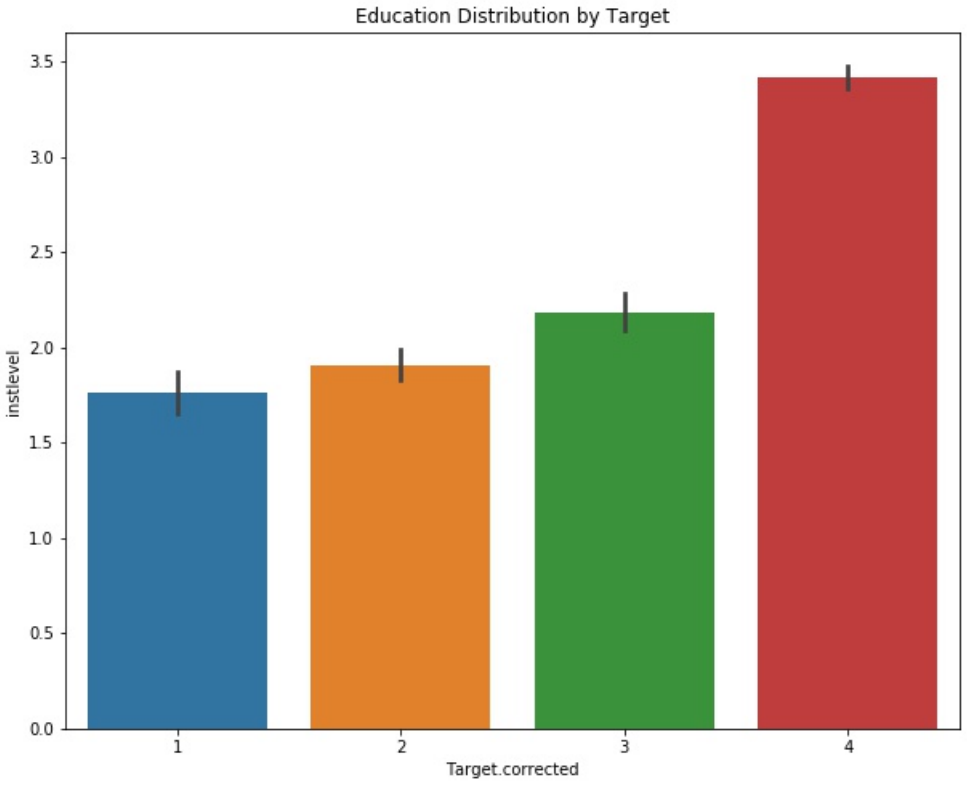

Figure 12: Distribution of education level over different categories of poverty.

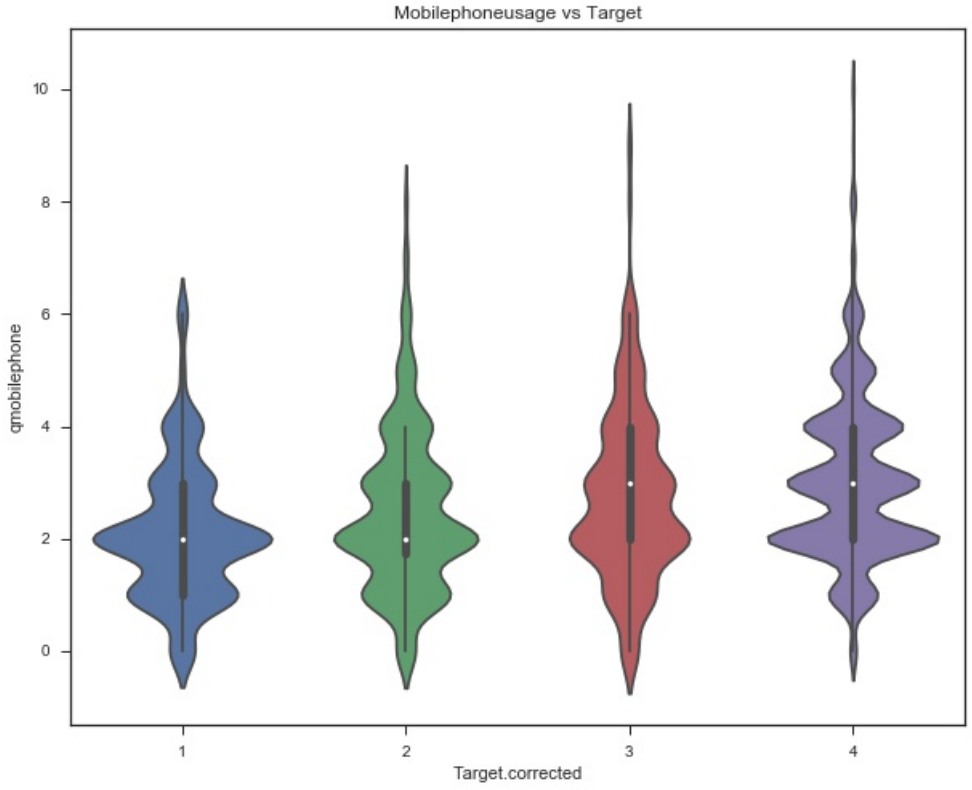

Figure 13: Violin plot of number of mobile phone owned by different poverty level.

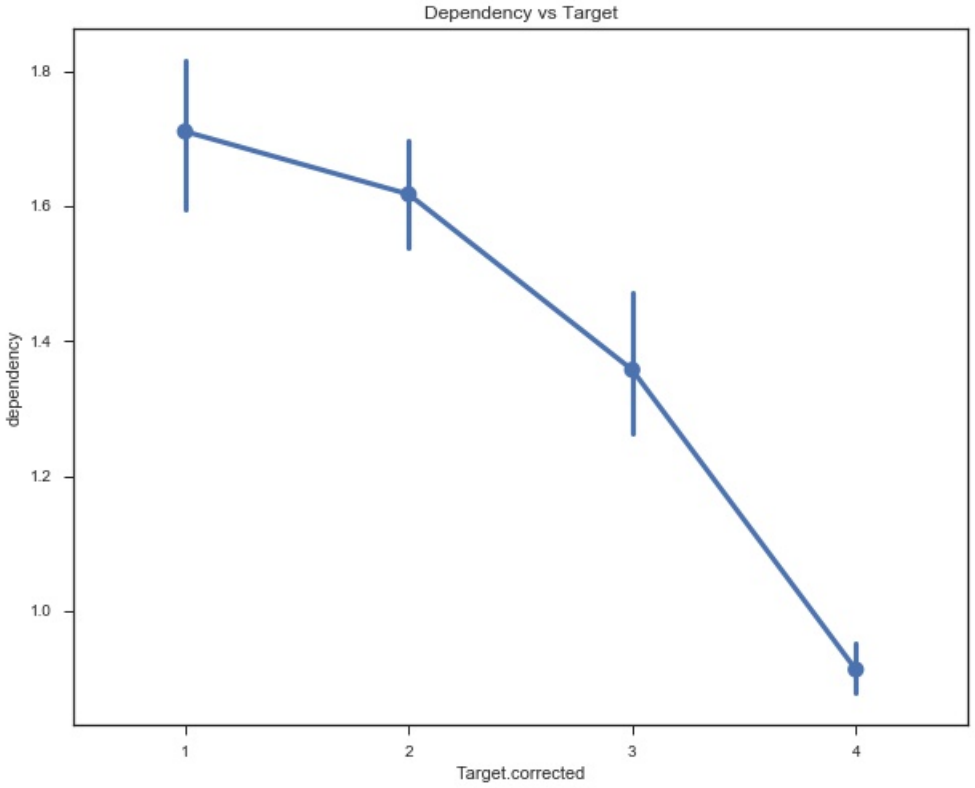

Figure 14: Point plot of dependencies by different poverty level.

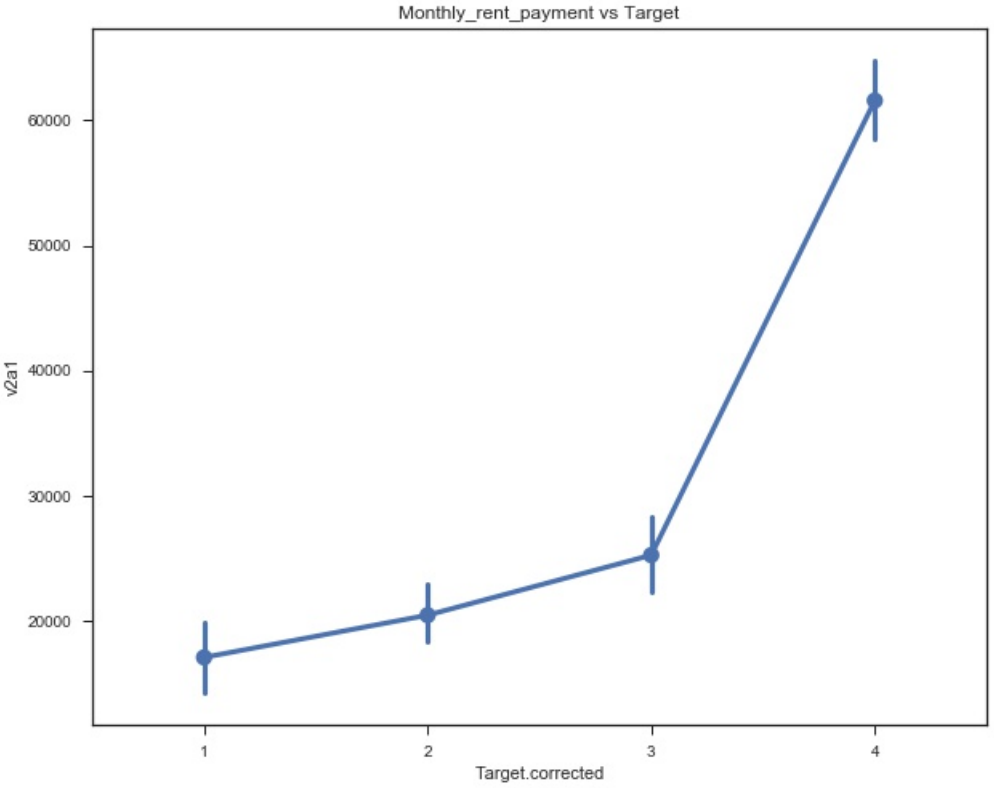

Figure 15: House rent paid by different poverty level.

7 Model Building

We tried eight different classifiers to classify the households into different poverty levels.The codes has been attached below.

1. Rpart
2. Bagging
3. Adaboost
4. Gradientboost

5. Naive Bayes classifier
6. Random Forest
7. XGBoost
8. Neural Network

```
In [32]: dataset = pd.read_csv('trainv1.csv')

         x = dataset.iloc[:,2:96]
         y = dataset.iloc[:,96]

         x_train ,x_test ,y_train,y_test = train_test_split(x,y,
                                                            test_size =0.25 ,
                                                            random_state = 0)
         target_names = ['extreme','moderate', 'vulnerable', 'non vulnerable']
```

Figure 16: Screenshot of python code for reading of dataset written in Jupyter notebook.

```
In [14]: from sklearn.ensemble import AdaBoostClassifier
         AdaBoost_model = AdaBoostClassifier(n_estimators=700, random_state=0)
         AdaBoost_model.fit(x_train, y_train)

         y_pred_ada = AdaBoost_model.predict(x_test)
         cm = confusion_matrix(y_test,y_pred_ada)
         print(cm)

         model_performance(cm)
         print(classification_report(y_test, y_pred_ada, target_names=target_names))
```

Figure 17: Screenshot of python code for model building with Ada Boost in Jupyter notebook.

```
In [16]: from sklearn.ensemble import RandomForestClassifier
         rf_model = RandomForestClassifier(n_estimators = 85,
                                           random_state = 0,
                                           max_features = 0.5)
         rf_model.fit(x_train, y_train)

         y_pred_rf = rf_model.predict(x_test)
         cm = confusion_matrix(y_test,y_pred_rf)
         print(cm)
         model_performance(cm)
         print(classification_report(y_test, y_pred_rf, target_names=target_names))
```

Figure 18: Screenshot of python code for model building with Random Forest in Jupyter notebook.

```
In [17]: from sklearn.ensemble import BaggingClassifier
         bagging_model = BaggingClassifier(n_estimators=30, random_state=0)
         bagging_model.fit(x_train, y_train)

         y_pred_bagging = bagging_model.predict(x_test)
         cm = confusion_matrix(y_test,y_pred_bagging)
         print(cm)
         model_performance(cm)
         print(classification_report(y_test, y_pred_bagging, target_names=target_names))
```

Figure 19: Screenshot of python code for model building with Bagging in Jupyter notebook.

In [34]:
```
from sklearn.tree import DecisionTreeClassifier

clf_gini = train_using_gini(x_train, y_train)
y_pred_DTgini = prediction(x_test, clf_gini)
cm = confusion_matrix(y_test,y_pred_DTgini)
print(cm)
model_performance(cm)
print(classification_report(y_test, y_pred_DTgini,
                            target_names=target_names))

clf_entropy = train_using_entropy(x_train, y_train)
y_pred_DTentropy = prediction(x_test, clf_entropy)
cm = confusion_matrix(y_test,y_pred_DTentropy)
print(cm)
model_performance(cm)
print(classification_report(y_test, y_pred_DTentropy,
                            target_names=target_names))
```

Figure 20: Screenshot of python code for model building with Decision Tree written in Jupyter notebook.

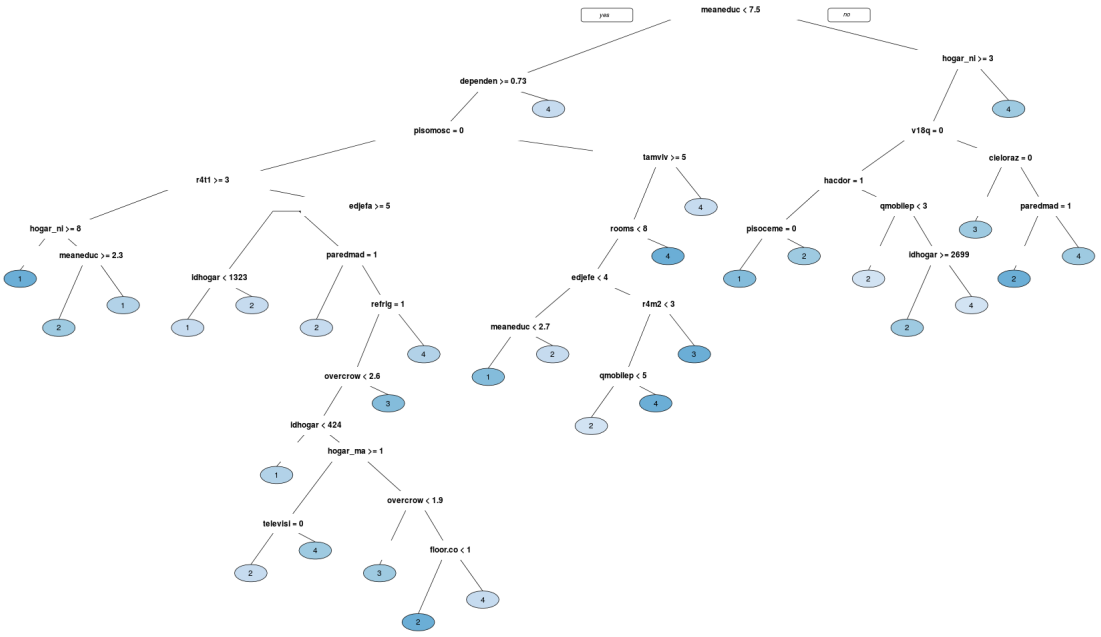

Figure 21: RPart tree built after training of the model using training set.

8 Results and Discussion

By comparing eight different models we got the highest accuracy as well as Fscore for Random Forest followed by Gradient Boost .So the best classification algorithm for the given data set is Random Forest .

```
In [35]: from xgboost import XGBClassifier

         XGB_model = XGBClassifier()
         XGB_model.fit(x_train, y_train)
         y_pred_XGB = XGB_model.predict(x_test)
         cm = confusion_matrix(y_test,y_pred_XGB)
         print(cm)
         model_performance(cm)
         print(classification_report(y_test, y_pred_XGB, target_names=target_names))
```

Figure 22: Screenshot of python code for model building with XG Boost written in Jupyter notebook.

```
In [10]: from sklearn.neighbors import KNeighborsClassifier
         knn_model = KNeighborsClassifier(n_neighbors = 4).fit(x_train, y_train)
         y_pred_KNN = knn_model.predict(x_test)

         cm = confusion_matrix(y_test, y_pred_KNN)
         print(cm)
         model_performance(cm)
         print(classification_report(y_test, y_pred_KNN, target_names=target_names))
```

Figure 23: Screenshot of python code for model building using KNN written in Jupyter notebook.

```
In [11]: from sklearn.naive_bayes import GaussianNB
         NB_model = GaussianNB().fit(x_train, y_train)
         y_pred_NB = NB_model.predict(x_test)

         cm = confusion_matrix(y_test, y_pred_NB)
         print(cm)
         model_performance(cm)
         print(classification_report(y_test, y_pred_NB, target_names=target_names))
```

Figure 24: Screenshot of python code for model building using Naive Bayes' Classifier written in Jupyter notebook.

```
In [11]: from sklearn.naive_bayes import GaussianNB
         NB_model = GaussianNB().fit(x_train, y_train)
         y_pred_NB = NB_model.predict(x_test)

         cm = confusion_matrix(y_test, y_pred_NB)
         print(cm)
         model_performance(cm)
         print(classification_report(y_test, y_pred_NB, target_names=target_names))
```

Figure 25: Screenshot of python code for model building using Naive Bayes' Classifier written in Jupyter notebook.

```
learning_rate = 0.001
epochs = 85
batch_size = 5

model = Sequential()
model.add(Dense(32, input_dim=94, activation='relu'))
model.add(Dense(4, activation='softmax'))
adam = keras.optimizers.Adam(lr=learning_rate, beta_1=0.9,
    beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
sgd = keras.optimizers.SGD(lr=learning_rate,
    decay=1e-6, momentum=0.9, nesterov=True)

model.compile(loss='categorical_crossentropy', optimizer=sgd,
    metrics=['accuracy'])
print(model.summary())

for e in range(epochs):
    print(e)
    model.fit( x_train, label_train, epochs= 1,
        batch_size = batch_size,verbose=0,shuffle= False)

pred_label = model.predict_classes(x_test)
pred_label = pred_label+1
test_accuracy = sum(pred_label==y_test)/ float(len(y_test))*100
print('Test accuracy : %.2f '%test_accuracy+'%')
```

Figure 26: Python code written in Sublime Text editor using Keras framework.

```
[[  76   48   21   44]
 [  51  153   51  159]
 [  27   43   80  160]
 [  55  102   70 1250]]
TP: 76
TN: 153
FP: 51
FN: 48
P: 124
N: 204
accuracy: 69.8170731707317
error: 30.182926829268293
sensitivity: 61.29032258064516
specificity: 75.0
                precision    recall  f1-score   support

       extreme       0.36      0.40      0.38       189
      moderate       0.44      0.37      0.40       414
    vulnerable       0.36      0.26      0.30       310
non vulnerable       0.77      0.85      0.81      1477

   avg / total       0.63      0.65      0.64      2390
```

Figure 27: Ada Boost Results

```
[[  61   42    2   84]
 [  17  184    2  211]
 [   6   67   35  202]
 [   4   47    0 1426]]
TP: 61
TN: 184
FP: 17
FN: 42
P: 103
N: 201
accuracy: 80.5921052631579
error: 19.407894736842106
sensitivity: 59.22330097087378
specificity: 91.54228855721394
                precision    recall  f1-score   support

       extreme       0.69      0.32      0.44       189
      moderate       0.54      0.44      0.49       414
    vulnerable       0.90      0.11      0.20       310
non vulnerable       0.74      0.97      0.84      1477

   avg / total       0.72      0.71      0.66      2390
```

Figure 28: XG Boost Results

```
[[ 160   15    4   10]
 [   8  376    5   25]
 [   3    8  276   23]
 [   6    8   14 1449]]
TP: 160
TN: 376
FP: 8
FN: 15
P: 175
N: 384
accuracy: 95.8855098389982
error: 4.114490161001789
sensitivity: 91.42857142857143
specificity: 97.91666666666666
                precision    recall  f1-score   support

       extreme       0.90      0.85      0.87       189
      moderate       0.92      0.91      0.92       414
    vulnerable       0.92      0.89      0.91       310
non vulnerable       0.96      0.98      0.97      1477

   avg / total       0.95      0.95      0.95      2390
```

Figure 29: Gradient Boost Results

```
[[ 156   11    1   21]
 [   6  377    9   22]
 [   1    8  269   32]
 [   1    1   11 1464]]
TP: 156
TN: 377
FP: 6
FN: 11
P: 167
N: 383
accuracy: 96.9090909090909
error: 3.090909090909091
sensitivity: 93.41317365269461
specificity: 98.43342036553526
                precision    recall  f1-score   support

       extreme       0.95      0.83      0.88       189
      moderate       0.95      0.91      0.93       414
    vulnerable       0.93      0.87      0.90       310
non vulnerable       0.95      0.99      0.97      1477

   avg / total       0.95      0.95      0.95      2390
```

Figure 30: Random Forest Result

```
[[ 154   12    0   23]
 [   6  376    6   26]
 [   2    7  275   26]
 [   2    4   10 1461]]
TP: 154
TN: 376
FP: 6
FN: 12
P: 166
N: 382
accuracy: 96.71532846715328
error: 3.2846715328467155
sensitivity: 92.7710843373494
specificity: 98.42931937172776
                precision    recall  f1-score   support

       extreme       0.94      0.81      0.87       189
      moderate       0.94      0.91      0.92       414
    vulnerable       0.95      0.89      0.92       310
non vulnerable       0.95      0.99      0.97      1477

   avg / total       0.95      0.95      0.95      2390
```

Figure 31: Result of bagging

```
[[  90   34    7   58]
 [  30  226   30  128]
 [  15   42  135  118]
 [  18   35   44 1380]]
TP: 90
TN: 226
FP: 30
FN: 34
P: 124
N: 256
accuracy: 83.15789473684211
error: 16.842105263157894
sensitivity: 72.58064516129032
specificity: 88.28125
                precision    recall  f1-score   support

       extreme       0.59      0.48      0.53       189
      moderate       0.67      0.55      0.60       414
    vulnerable       0.62      0.44      0.51       310
non vulnerable       0.82      0.93      0.87      1477

   avg / total       0.75      0.77      0.75      2390
```

Figure 32: Result of RpartGini

```
[[ 132   20    0   37]
 [  48  251   16   99]
 [  26   42  138  104]
 [  24   51   29 1373]]
TP: 132
TN: 251
FP: 48
FN: 20
P: 152
N: 299
accuracy: 84.92239467849224
error: 15.077605321507761
sensitivity: 86.8421052631579
specificity: 83.94648829431438
                precision    recall  f1-score   support

       extreme       0.57      0.70      0.63       189
      moderate       0.69      0.61      0.65       414
    vulnerable       0.75      0.45      0.56       310
non vulnerable       0.85      0.93      0.89      1477

   avg / total       0.79      0.79      0.78      2390
```

Figure 33: Result of RpartEntropy

```
[[  16  111    3   59]
 [  10  255   20  129]
 [   4  146   21  139]
 [   5  283   73 1116]]
TP: 16
TN: 255
FP: 10
FN: 111
P: 127
N: 265
accuracy: 69.13265306122449
error: 30.86734693877551
sensitivity: 12.598425196850393
specificity: 96.22641509433963
                precision    recall  f1-score   support

       extreme       0.46      0.08      0.14       189
      moderate       0.32      0.62      0.42       414
    vulnerable       0.18      0.07      0.10       310
non vulnerable       0.77      0.76      0.76      1477

   avg / total       0.59      0.59      0.57      2390
```

Figure 34: Result of Naive Bayes algorithm.

```
[[  73   10    4  102]
 [ 109   59   21  225]
 [  43   19   31  217]
 [  56    9   10 1402]]
TP: 73
TN: 59
FP: 109
FN: 10
P: 83
N: 168
accuracy: 52.589641434262944
error: 47.410358565737056
sensitivity: 87.95180722891565
specificity: 35.11904761904761
                precision    recall  f1-score   support

       extreme       0.26      0.39      0.31       189
      moderate       0.61      0.14      0.23       414
    vulnerable       0.47      0.10      0.16       310
non vulnerable       0.72      0.95      0.82      1477

   avg / total       0.63      0.65      0.59      2390
```

Figure 35: Result of Neural Network

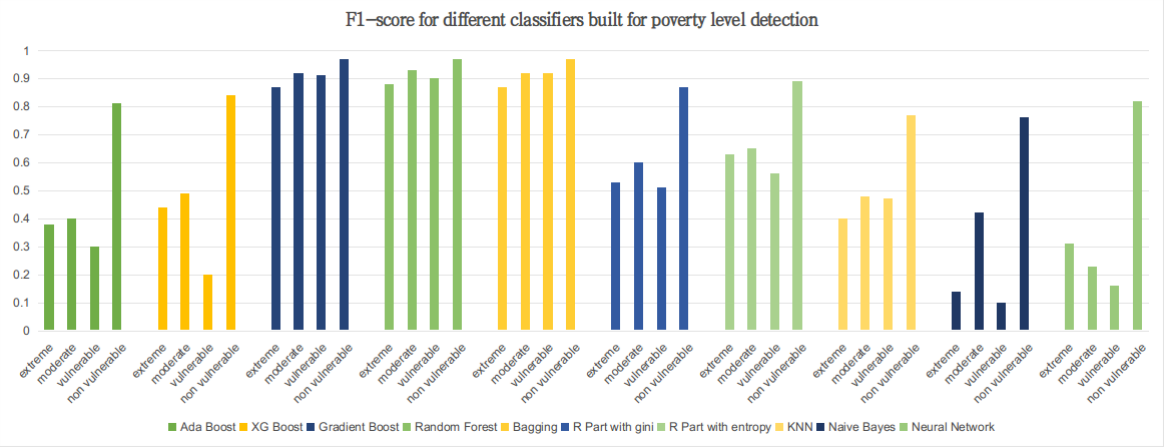

Figure 36: F score comparing all the models