Machine Learning Engineer Nanodegree

Capstone Project

Abhishek Mishra
February 4th, 2019

I. Definition

Project Overview

Many social programs have a hard time making sure the right people are given enough aid. It's especially tricky when a program focuses on the poorest segment of the population. The world's poorest typically can't provide the necessary income and expense records to prove that they qualify.

In Latin America, one popular method uses an algorithm to verify income qualification. It's called the Proxy Means Test (or PMT). With PMT, agencies use a model that considers a family's observable household attributes like the material of their walls and ceiling, or the assets found in the home to classify them and predict their level of need.

To improve on PMT, the IDB (the largest source of development financing for Latin America and the Caribbean) has turned to the Kaggle community. They believe that new methods beyond traditional econometrics, based on a dataset of Costa Rican household characteristics, might help improve PMT's performance.

Beyond Costa Rica, many countries face this same problem of inaccurately assessing social need. If we as Data scientists can generate an improvement, the new algorithm could be implemented in other countries around the world for social welfare.

Multiclass classification algorithms have been quite popular in academics and industry to solve real world problems and there are several papers published on this. We can refer below paper which depicts classification algorithms and results obtained on various data sets.

https://arxiv.org/ftp/arxiv/papers/1609/1609.00843.pdf

Problem Statement

The Inter-American Development Bank is asking for algorithmic solutions, based on a dataset of Costa Rican household characteristics, with income qualification for some of the world's poorest families. It hosted dataset and relevant details on Kaggle platform to get the best solutions for classification of incomes. Below is the link of Kaggle competition hosted by IDB.

https://www.kaggle.com/c/costa-rican-household-poverty-prediction

Here we have to classify income of families in 4 groups: 1(extreme poverty) ,2 (moderate poverty) ,3(vulnerable households) ,4(non-vulnerable households) based on household characteristics (dependent variables) hence this problem can be solved using supervised multi-class classifications algorithms.

To solve this problem, we will use few very popular and efficient supervised learning algorithm and train given dataset, but in real world data is not always handy and we might have to go through few data pre-processing steps which typically includes below:

- Handling missing values
- Treating outliers
- Transforming skewed continuous features
- Normalizing numerical features
- Encoding categorical features (Label encoding or one-hot encoding)

In addition to it, one should be familiar with data as it helps choosing right model and relevant features for our problem. We will be doing exploratory data analysis to get insights from data and relationship among features and with target variable. We might drop some of the features those are highly correlated to other features. EDA also helps significantly in following data pre-processing steps.

Once we obtain clean dataset after EDA and pre-processing, we will split our data set into training/validation set and start training it using different classification algorithms.

The following are some of the supervised learning models that we might use to model our dataset and pick specific learner which achieves best score.

- Gaussian Naive Bayes (GaussianNB)
- Decision Trees
- Ensemble Methods (Bagging, AdaBoost, Random Forest, Gradient Boosting)
- K-Nearest Neighbors (KNeighbors)
- Stochastic Gradient Descent Classifier (SGDC)
- Support Vector Machines (SVM)
- Logistic Regression

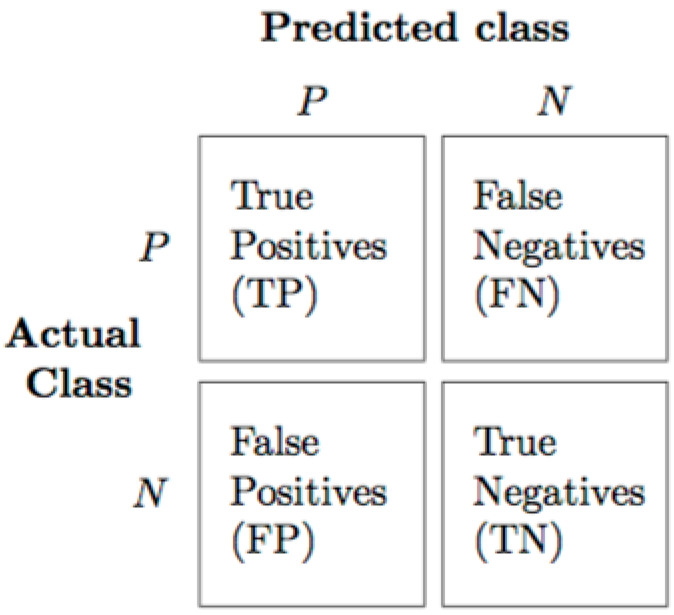
Evaluation Metrics

Submission will be evaluated based on macro F1 score.

References used: https://sebastianraschka.com/faq/docs/multiclass-metric.html
https://en.wikipedia.org/wiki/F1_score

In statistical analysis of binary classification, the F1 score (also F-score or F-measure) is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score: p is the number of correct positive results divided by the number of all positive results returned by the classifier, and r is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

We can generalize all the binary performance metrics such as precision, recall, and F1-score etc. to multi-class settings. In the binary case, we have



		Predicted class	
		P	N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

$$PRE = \frac{TP}{TP + FP}$$

$$REC = TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$

$$F_1 = 2 \cdot \frac{PRE \cdot REC}{PRE + REC}$$

(PRE=precision, REC=recall, F1=F1-Score) And to generalize this to multi-class, assuming we have a One-vs-All (OvA) classifier, we can either go with the “micro” average or the “macro” average. In “micro averaging,” we’d calculate the performance, e.g., precision, from the individual true positives, true negatives, false positives, and false negatives of the the k-class model:

$$PRE_{micro} = \frac{TP_1 + \cdots + TP_k}{TP_1 + \cdots + TP_k + FP_1 + \cdots + FP_k}$$

And in macro-averaging, we average the performances of each individual class:

$$PRE_{macro} = \frac{PRE_1 + \cdots + PRE_k}{k}$$

$$PRE_{macro} = \frac{PRE_1 + \cdots + PRE_k}{k}$$

II. Analysis

Data Exploration

Dataset for this problem can be accessed thorough Kaggle link:

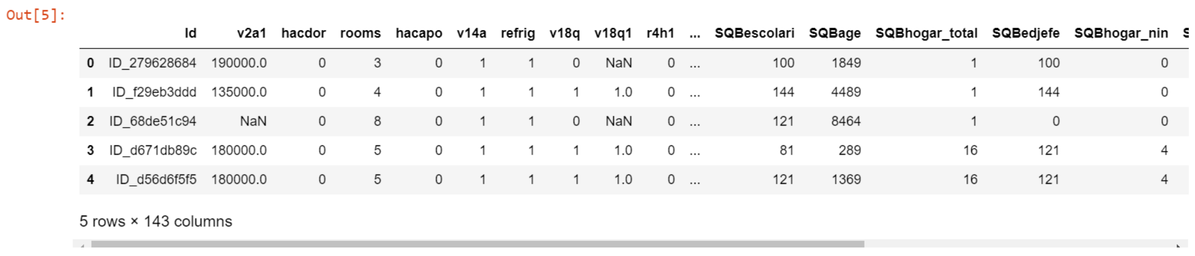
https://www.kaggle.com/c/costa-rican-household-poverty-prediction/data

{train|test}.csv - the training set

- This is the main table, broken into two files for Train (with a Target column) and Test (without the Target column).
- One row represents one person in our data sample.
- Multiple people can be part of a single household. Only predictions for heads of household are scored.
- Total no of training records=9558
- Total no of features=142
- Total no of test records=23.9k
- Class distribution: Classes are not well balanced as 62.7% examples belong to category:4 and rest 31.3% examples belong to other 3 classes.

Target	No. of examples
1	755
2	1597
3	1209
4	5996

Sample of dataset:



Out[5]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolari	SQBage	SQBhogar_total	SQBedjefe	SQBhogar_nin	S
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	...	100	1849	1	100	0	
1	ID_f29eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	...	144	4489	1	144	0	
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	...	121	8464	1	0	0	
3	ID_d671db89c	180000.0	0	5	0	1	1	1	1.0	0	...	81	289	16	121	4	
4	ID_d56d6f5f5	180000.0	0	5	0	1	1	1	1.0	0	...	121	1369	16	121	4	

5 rows × 143 columns

Core Data fields

- Id - a unique identifier for each row.
- Target - the target is an ordinal variable indicating groups of income levels.
  1 = extreme poverty
  2 = moderate poverty
  3 = vulnerable households
  4 = non-vulnerable households

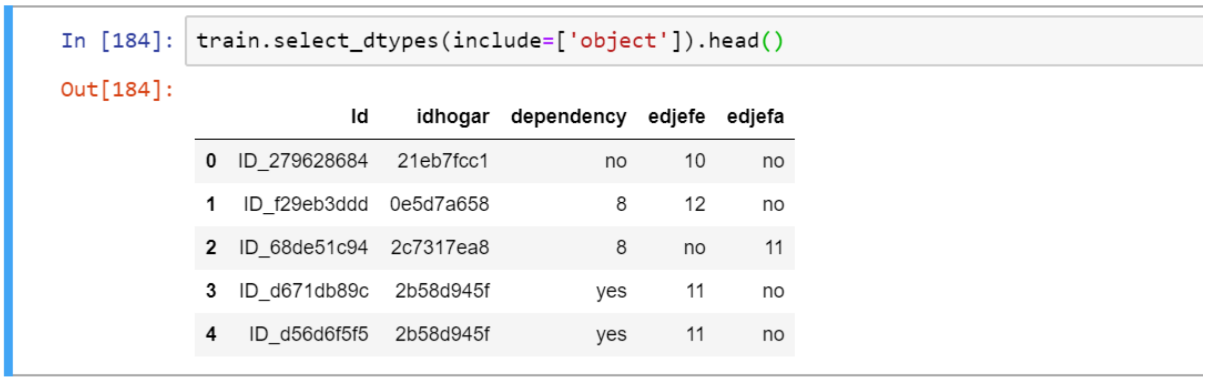
- idhogar - this is a unique identifier for each household. This can be used to create household-wide features, etc. All rows in a given household will have a matching value for this identifier.
- parentesco1 - indicates if this person is the head of the household.
- This data contains 142 total columns.

Below are some snippets of data exploration:

In [4]: train.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9557 entries, 0 to 9556
Columns: 143 entries, Id to Target
dtypes: float64(8), int64(130), object(5)
memory usage: 10.4+ MB

Analyze columns with string values:

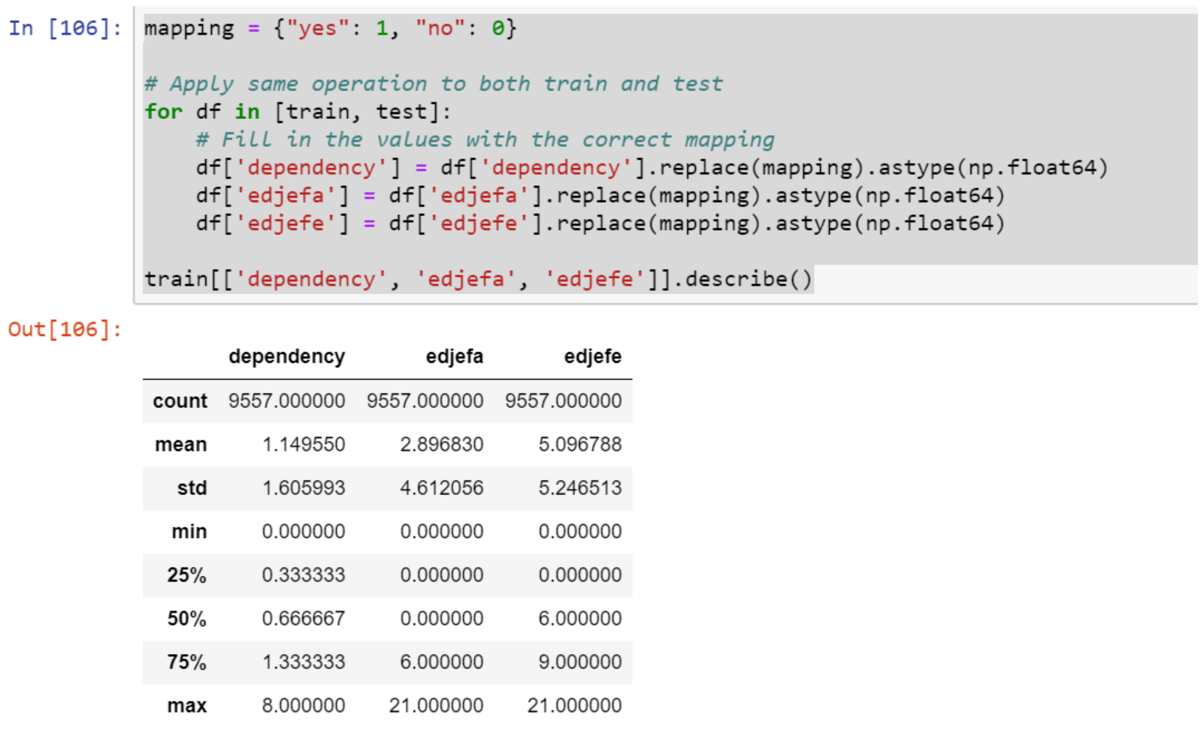
In [7]: train.select_dtypes(include=['object']).head()



```
In [184]: train.select_dtypes(include=['object']).head()
Out[184]:
```

	Id	idhogar	dependency	edjefe	edjefa
0	ID_279628684	21eb7fcc1	no	10	no
1	ID_f29eb3ddd	0e5d7a658	8	12	no
2	ID_68de51c94	2c7317ea8	8	no	11
3	ID_d671db89c	2b58d945f	yes	11	no
4	ID_d56d6f5f5	2b58d945f	yes	11	no

Idhogar : Unique identifier for house hold
Below three features contains mixed values (numbers and yes ,no).We can convert yes no into corresponding numeric values so that we can convert these columns into float type.
dependency: Dependency rate, calculated = (number of members of the household younger than 19 or older than 64)/(number of member of household between 19 and 64)
edjefe: years of education of male head of household, based on the interaction of escolari (years of education), head of household and gender, yes=1 and no=0
edjefa: years of education of female head of household, based on the interaction of escolari (years of education), head of household and gender, yes=1 and no=0
We see that for columns dependency,edjefa,edjefe values are mixed of boolean and numbers.
We should convert boolean into numbers according to above description.



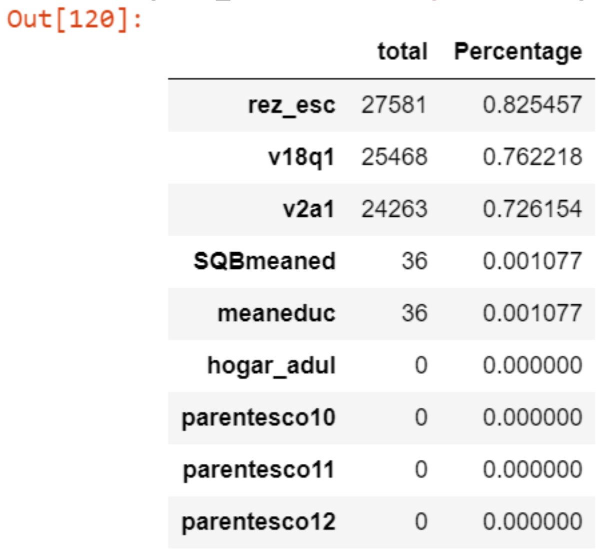
```
In [106]: mapping = {"yes": 1, "no": 0}

          # Apply same operation to both train and test
          for df in [train, test]:
              # Fill in the values with the correct mapping
              df['dependency'] = df['dependency'].replace(mapping).astype(np.float64)
              df['edjefa'] = df['edjefa'].replace(mapping).astype(np.float64)
              df['edjefe'] = df['edjefe'].replace(mapping).astype(np.float64)

          train[['dependency', 'edjefa', 'edjefe']].describe()
```

Out[106]:

	dependency	edjefa	edjefe
count	9557.000000	9557.000000	9557.000000
mean	1.149550	2.896830	5.096788
std	1.605993	4.612056	5.246513
min	0.000000	0.000000	0.000000
25%	0.333333	0.000000	0.000000
50%	0.666667	0.000000	6.000000
75%	1.333333	6.000000	9.000000
max	8.000000	21.000000	21.000000

Identifying missing values

```
In [23]: missing=pd.DataFrame(data.isnull().sum()).rename(columns={0:'total'})
missing['Percentage']=missing['total']/len(data)
In [24]: missing.sort_values('Percentage',ascending=False).head(10).drop('Target')
```

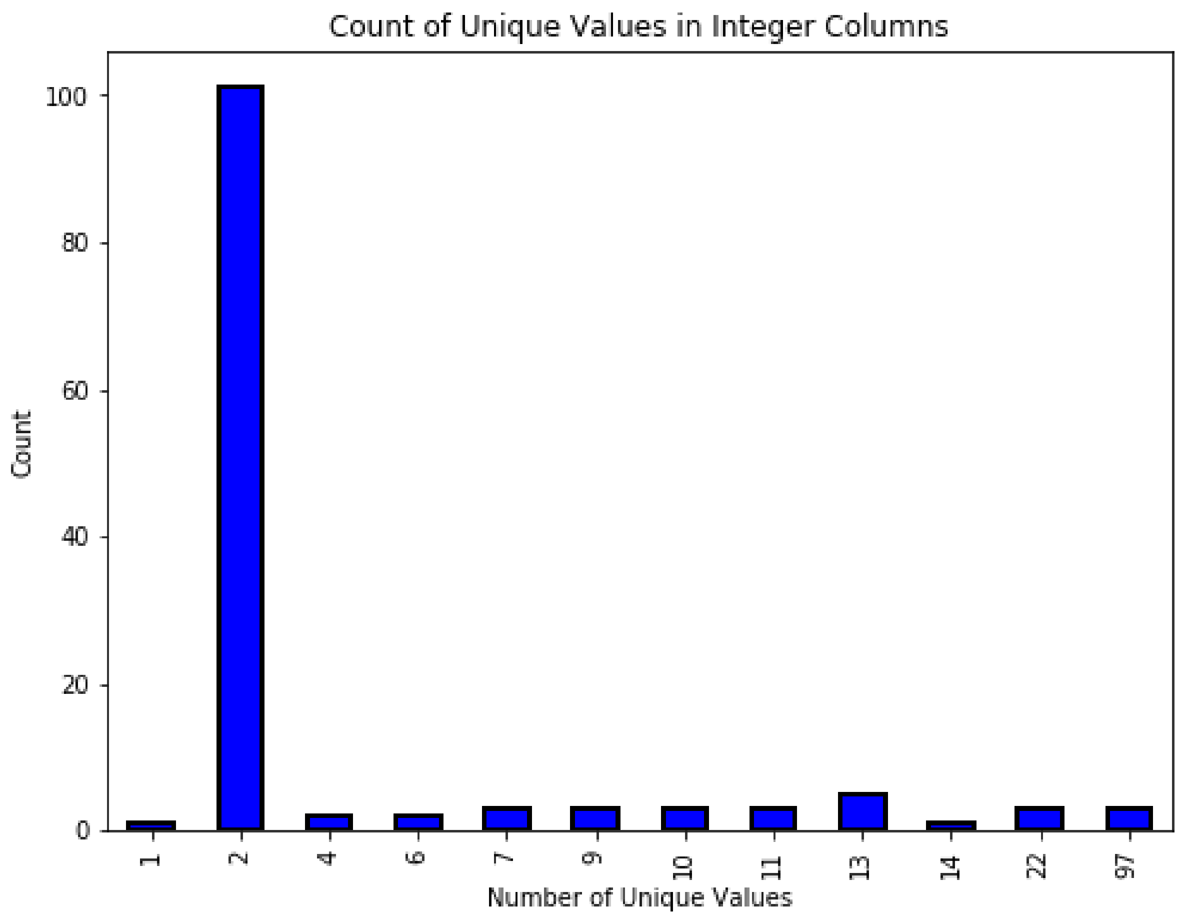


Out[120]:

	total	Percentage
rez_esc	27581	0.825457
v18q1	25468	0.762218
v2a1	24263	0.726154
SQBmeaned	36	0.001077
meaneduc	36	0.001077
hogar_adul	0	0.000000
parentesco10	0	0.000000
parentesco11	0	0.000000
parentesco12	0	0.000000

Analyzing distribution of numerical columns:

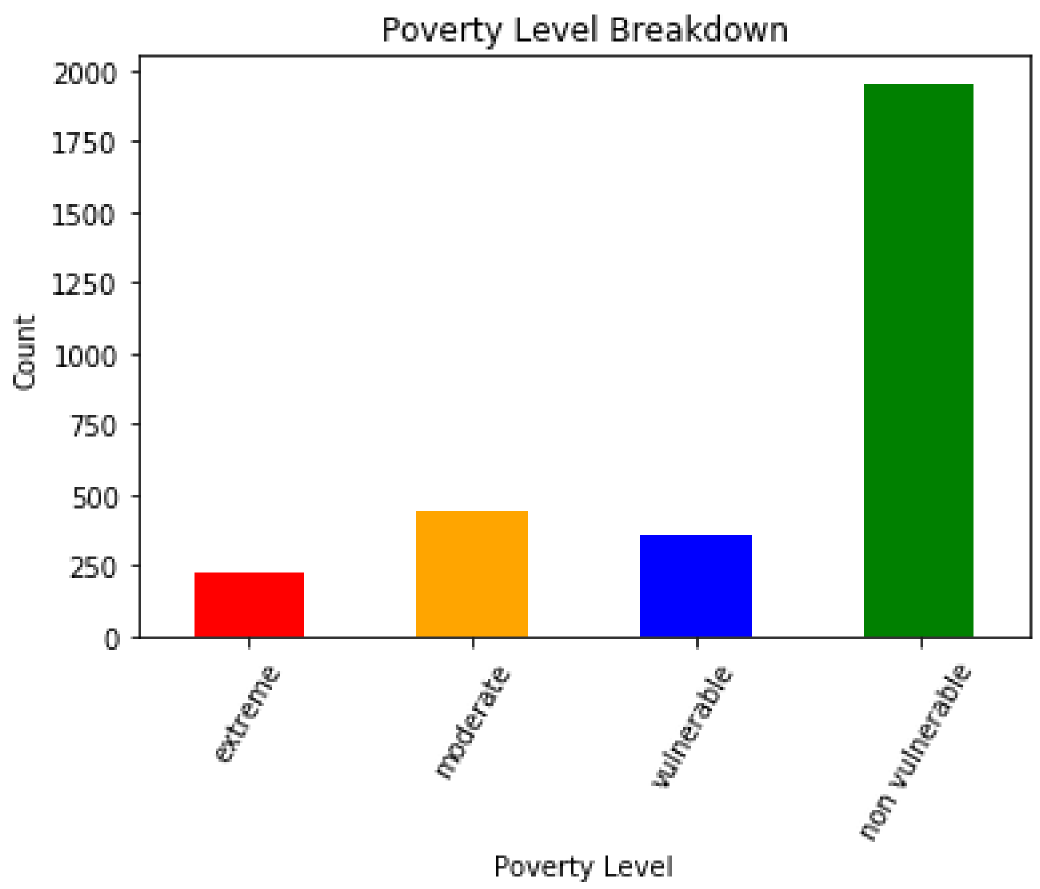
```
In [5]: train.select_dtypes(include=['int64']).nunique().value_counts().sort_index().plot.bar(color
figsize = (8, edgecolor = plt.xlabel('Number of Unique Values'); plt.ylabel('Count');
plt.title('Count of Unique Values in Integer Columns');
```



Exploratory Visualization

Below are some snippets of data visualization:

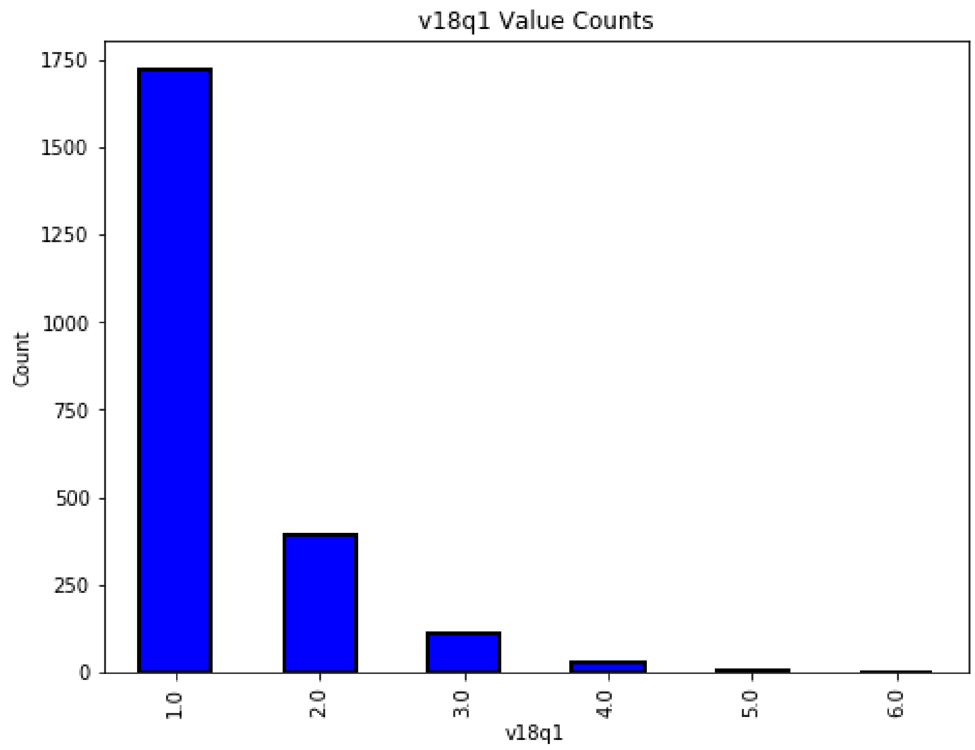
```
In [16]: from collections import OrderedDict
# Color mapping
colors = OrderedDict({1: 'red', 2: 'orange', 3: 'blue', 4: 'green'})
poverty_mapping = OrderedDict({1: 'extreme', 2: 'moderate', 3: 'vulnerable', 4: 'non vulnerable'})

In [18]: label_counts.plot.bar(color = colors.values())
plt.xlabel('Poverty Level');
plt.ylabel('Count');
plt.xticks([x - 1 for x in poverty_mapping.keys()],
list(poverty_mapping.values()), rotation = 60)
plt.title('Poverty Level Breakdown');
```



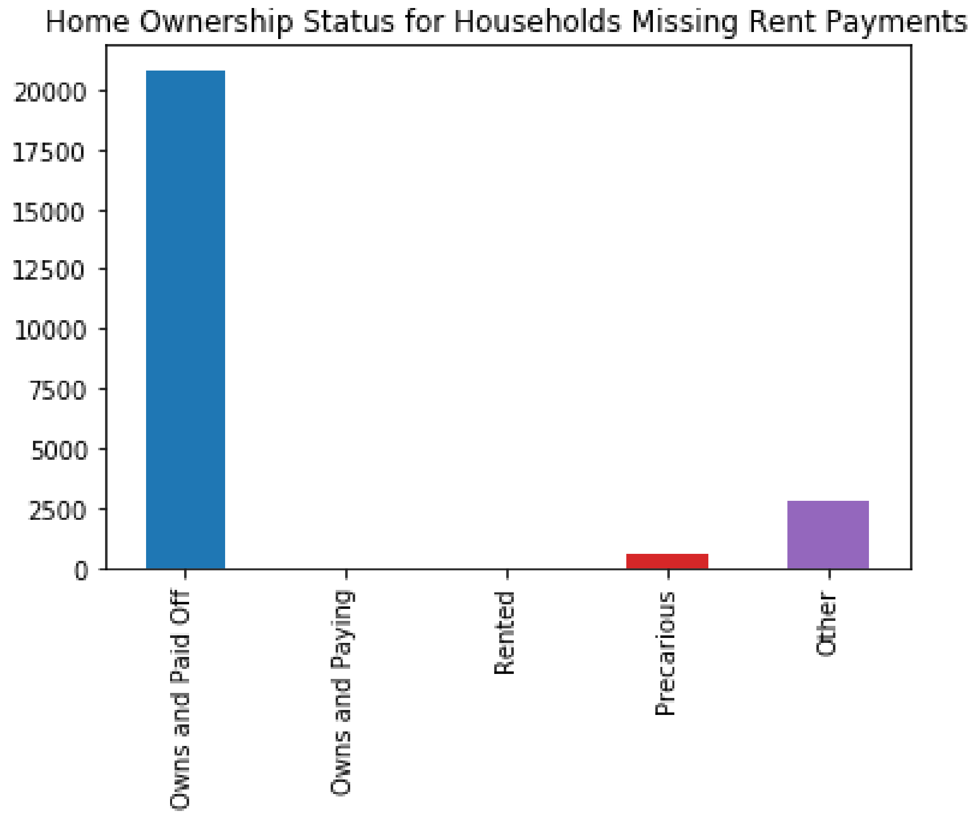
Defined a function plot_value_counts to display count distinct values of features:

```
In [25]: def plot_value_counts(df, col, heads_only = False):
"""Plot value counts of a column, optionally with only the heads of a household"""
# Select heads of household
if heads_only:
df = df.loc[df['parentesco1'] == 1].copy()
plt.figure(figsize = (8, 6))
df[col].value_counts().sort_index().plot.bar(color = 'blue',
edgecolor = 'k',
linewidth = 2)
plt.xlabel(f'{col}'); plt.title(f'{col} Value Counts'); plt.ylabel('Count')
plt.show();
```

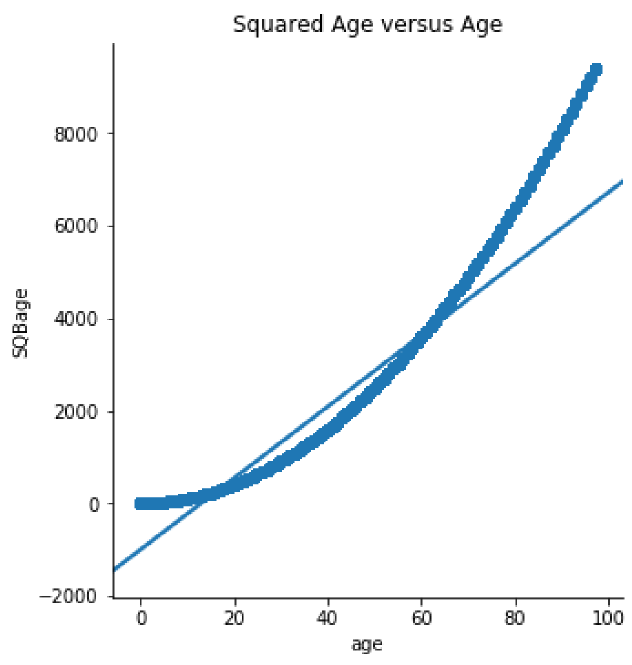
Lets plot for feature v18q1: number of tablets household owns. Since its household level feature ,we will only look for counts related to head of household



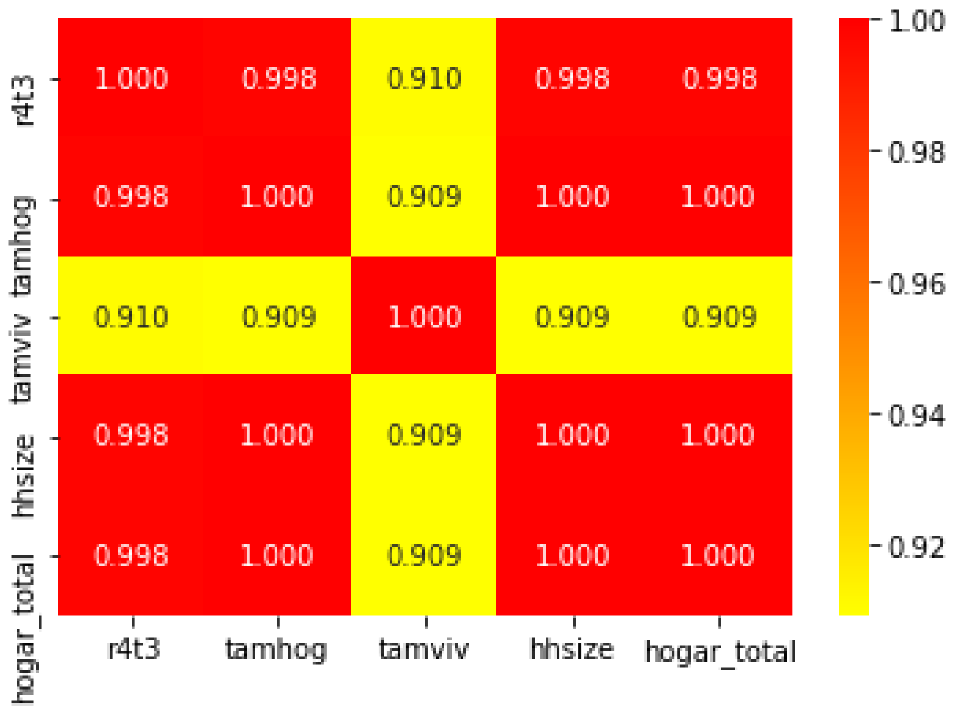
Lets plot v2a1: Monthly rent payment feature to visualize different categories.



We have few squared features in dataset .We can visualize relationship of feature and corresponding squared feature and might want to delete it if both are corelated to each other.

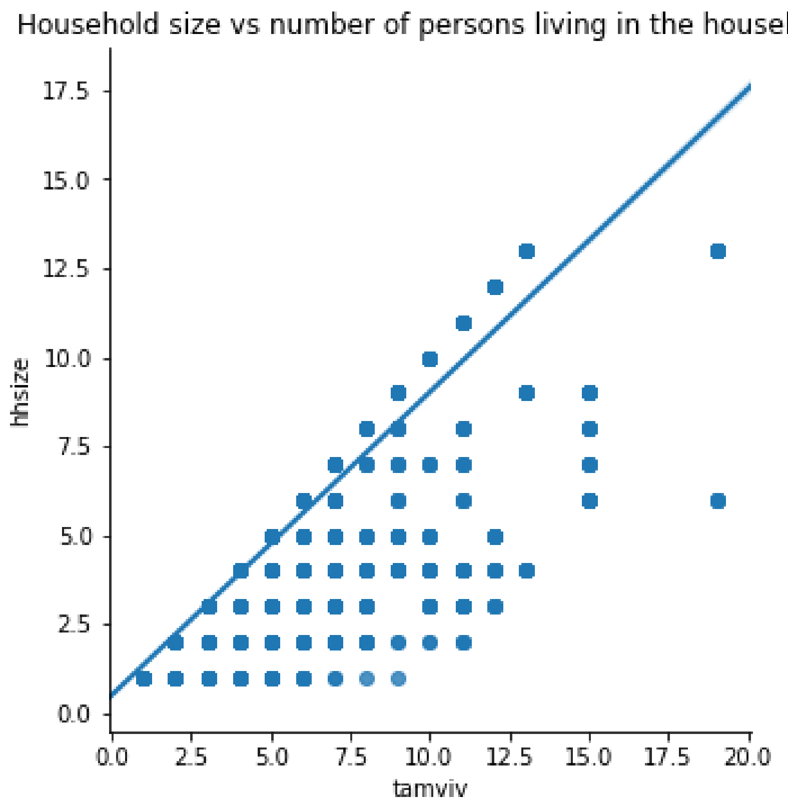


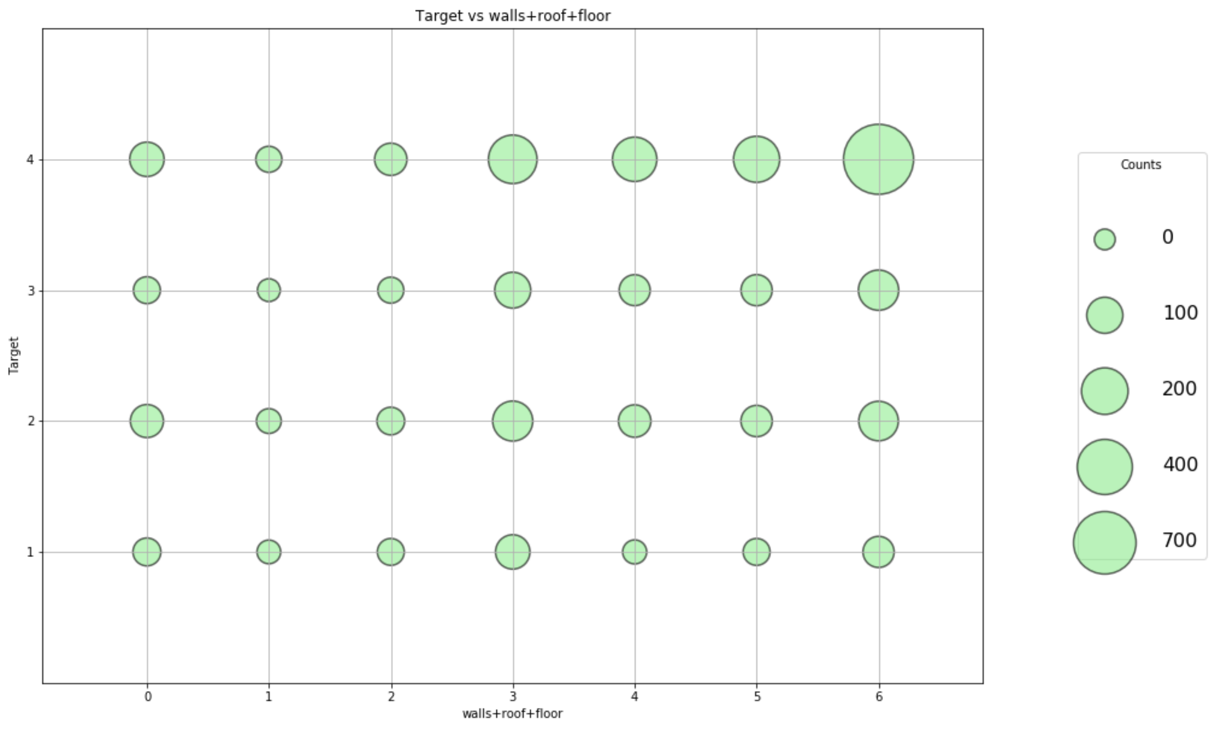
Used heatmap to reflect highly corelated features.



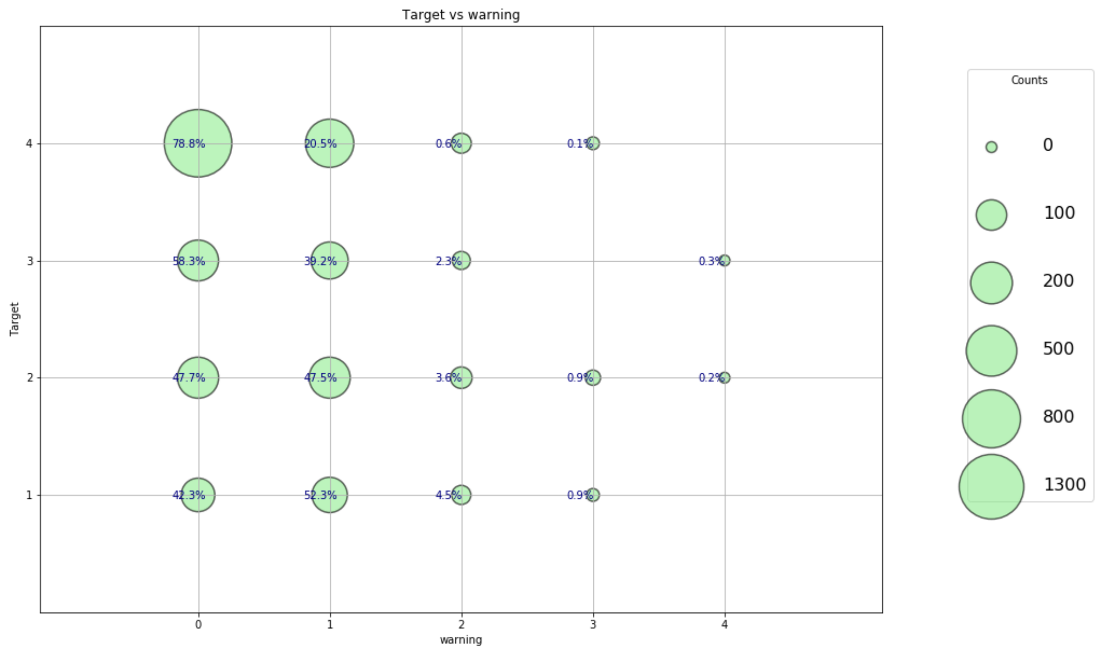
	r4t3	tamhog	tamviv	hhsize	hogar_total
r4t3	1.000	0.998	0.910	0.998	0.998
tamhog	0.998	1.000	0.909	1.000	1.000
tamviv	0.910	0.909	1.000	0.909	0.909
hhsize	0.998	1.000	0.909	1.000	1.000
hogar_total	0.998	1.000	0.909	1.000	1.000

Some other useful and informative visualizations are:

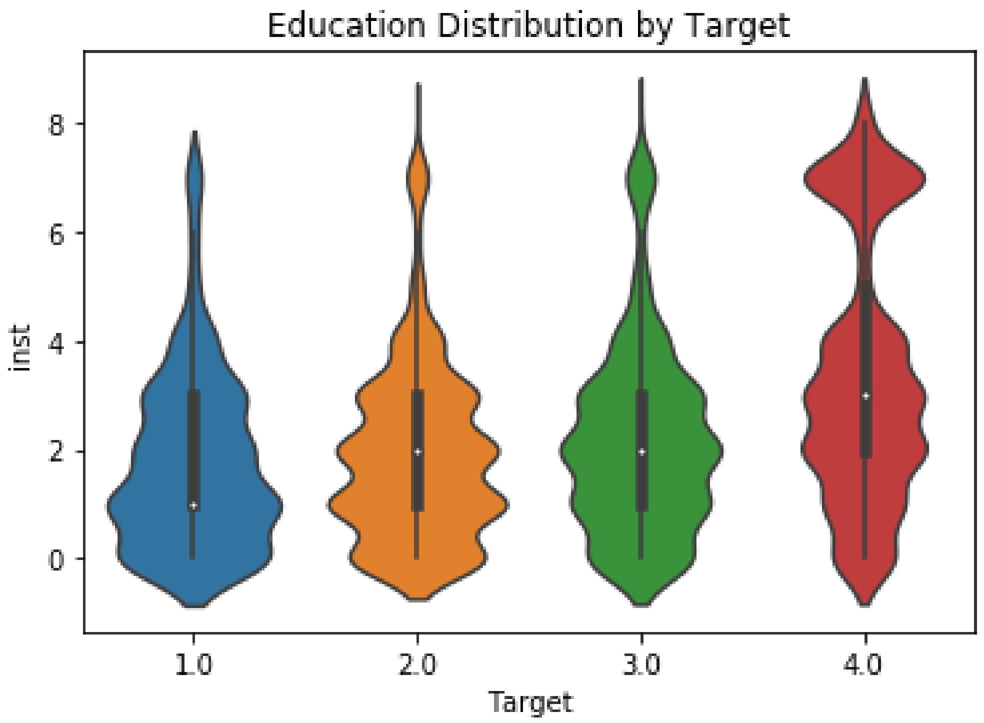




* Size represents raw count while % is for a given y value.



* Size represents raw count while % is for a given y value.



Reference used: Kaggle discussions and feature analysis by Will Koehrsen

Algorithms and Techniques

Below is the summary of supervised learning algorithms we will use to implement our solution:

DecisionTree: Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. In general decision tree classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on classification.

Strengths of the model: Simple to understand and interpret. It can be combined with other decision techniques.Decision trees implicitly perform variable screening or feature selection.Decision trees require relatively little effort from users for data preparation(Normalization,scaling etc.).Decision trees do not require any assumptions of linearity in the data,therefore nonlinear relationships between parameters do not affect tree performance.

Ada Boost: is short for Adaptive Boosting. It is basically a machine learning algorithm that is used as a classifier. Whenever you have a large amount of data and you want divide it into different categories, we need a good classification algorithm to do it. We usually use AdaBoost in conjunction with other learning algorithms to improve their performance. Hence the word 'boosting', as in it boosts other algorithms! Boosting is a general method for improving the accuracy of any given learning algorithm. So obviously, adaptive boosting refers to a boosting algorithm that can adjust itself to changing scenarios.

AdaBoost was formulated by Yoav Freund and Robert Schapire. Problems in machine learning tend to suffer from the curse of dimensionality. What it means is that a single feature point ends up having a huge dimensionality. As in, the feature vector used to describe a particular thing can be very huge. So if each sample consists of a huge number of potential features, the overall system becomes very slow. This is the reason we cannot use a powerful model with a full feature set because it cannot run in real time. We can only afford to have simple machine learning algorithms. But if the algorithms are too simple, they tend to be less accurate. They are called 'weak learners'. So we cascade them together to create a strong classifier. The output of 'weak learners' is combined into a weighted sum that represents the final output of the boosted classifier.

How is it "adaptive"?

AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. When we collect the set of weak learners, we give equal weights to all of them. But on each round, the weights of incorrectly classified examples are increased so that the weak learner is forced to focus on the hard examples in the training set. If you consider our earlier example of classifying people in San Francisco, it's like saying that you trust all three people equally. But with each passing day, you realize that the second person is more accurate than the other two. So you give more weightage to his guesses and less weightage to the other two. This way, the overall accuracy will increase. Basically, the weak learner's job is the find a weak hypothesis that is appropriate for the current distribution of the data.

References: http://mccormickml.com/2013/12/13/adaboost-tutorial/
https://prateekvjoshi.com/2014/05/05/what-is-adaboost/

Random Forest: Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates 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.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set X = x1, ..., xn with responses Y = y1, ..., yn, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For *b* = 1, ..., *B*:

1. Sample, with replacement, *n* training examples from *X*, *Y*; call these X_b, Y_b.
2. Train a classification or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x' or by taking the majority vote in the case of classification.

Reference: https://en.wikipedia.org/wiki/Random_forest

To properly evaluate the performance of each model I've chosen, it's important to create a training and predicting pipeline that allows you to quickly and effectively train models using various sizes (1%, 10%, and 100%) of training data and perform predictions on the testing data. While implementing pipeline, I will follow below steps:

- Import `f1_score` and `accuracy_score` from `sklearn.metrics`.
- Take 3 samples of training data (1%, 10%, and 100%)
- Create learners of chosen algorithms
- Fit the learner to the sampled training data and record the training time.
- Perform predictions on the test data `X_test`, and also on the first 300 training points `X_train[:300]`.Here selection of 300 is subjective and it might be developers choice.
    - Record the total prediction time.
- Calculate the accuracy score for both the training subset and testing set.
- Calculate the F-score for both the training subset and testing set.
- Visualize the results of pipeline execution for all learners to select the best performing learner/baseline model.

Benchmark Model

First, I trained out of the box random forest model and obtained F1 score, which is going to be threshold score and model would be benchmark model.

F1 score: 0.34
accuracy_score : 0.6436

III. Methodology

Data Preprocessing

For this problem I have followed couple of data pre-processing steps which is listed as below.

- Few features: dependency, edjefa, edjefe had mixed of boolean and integer values. I converted Boolean values into numbers (0,1) by using feature description provided in problem statement.
- Corrected labels for few of the records. There might be chances of individuals of same household having different poverty levels though it must be same at household level. I have identified such records and corrected respective labels based on label of house-head.
- Missing values: There were few features with missing values : rez_esc (years behind in school), v18q1(number of tablets household owns), v2a1 Monthly rent payment .to handle missing values, I have taken other related features and filled missing values based on that.As example:

  Missing value column ,v2a1: Monthly rent payment
  I considered below related variables as well to conclude on missing data for this column.
  tipovivi1, =1 own and fully paid house
  tipovivi2, "=1 own, paying in installments"
  tipovivi3, =1 rented
  tipovivi4, =1 precarious
  tipovivi5, "=1 other(assigned, borrowed)"

  Based on these columns, We can fill in households that own the house with 0 rent payment.I have applied similar logic for other columns with missing values. After following these steps, for remaining missing values,I have replaced with mean of features.

- There were few columns which reflects same information with some inherent order in it.I have clubbed all those features and created respective ordinal feature for them as example:

  Took these boolean features and converted to ordinal feature **inst**
  - instlevel1, =1 no level of education
  - instlevel2, =1 incomplete primary
  - instlevel3, =1 complete primary
  - instlevel4, =1 incomplete academic secondary level
  - instlevel5, =1 complete academic secondary level
  - instlevel6, =1 incomplete technical secondary level
  - instlevel7, =1 complete technical secondary level
  - instlevel8, =1 undergraduate and higher education
  - instlevel9, =1 postgraduate higher education

  ```
  ind['inst'] = np.argmax(np.array(ind[[c for c in ind if c.startswith('instl')]]), axis = 1)
  ```

- Aggregated individual level features into household level using aggregate functions as problem statement mentioned that only household level is considered for evaluation

- Normalized numerical features using minmax scaler before passing data to learning algorithms.

Implementation

After following data exploration visualization and pre-processing steps, it comes to modelling part.

In this project, I have employed several supervised algorithms to accurately model household's income levels using data of Costa Rican household characteristics. I then chose the best candidate algorithm from preliminary results and further optimize this algorithm to best model the data. Our goal with this implementation is to construct a model that accurately classifies income levels of families in given 4 categories.

Since its supervised learning problem, we need to find relationship of target variable with 142 features (dependent variables) using training set in such a way that model could be generalized to unseen data. As no of features are slightly high, I tried to eliminated unimportant features using feature selection methods and trained model with reduced feature space.

To properly evaluate the performance of each model I have choosen ,I created a training and predicting pipeline that allowed me to quickly and effectively train models using various sizes of training data and perform predictions on the testing data. We evaluated performance using evaluation metrics for classification algorithms: f-score, accuracy. Following execution of pipeline for different models, I picked the best-performing model and tried tuning its hyper-parameters to achieve best results.

List of algorithms used:

- Decision tree
- Random Forest
- Adaboost

I have already described these algorithms and high level steps of pipeline in **algorithms and techniques** section of this document.

Refinement

As we already trained out of the box random forest model and obtained F1 score, which is going to be threshold score, goal was to improve existing model or choose model with better score.

To achieve this, I created train_predict pipeline and trained three different classifiers(decisiontree,adaboost and random forest) and stored performance metrics in a dictionary. Now upon comparing performances of all models, I observed that Adaboost classifer performed well compare to other two models.

Then I decided to choose adaboost over out of the box random forest and perform hyper parameter tuning using grid search. I applied grid search on two hyperparameters:

```
parameters = {
    'n_estimators': [40,50,60,70,80],
    'learning_rate':[0.1,0.5, 1],
    }
```

n_estimators: The maximum number of estimators at which boosting is terminated). In case of perfect fit, the learning procedure is stopped early. In other words the number of models to iteratively train.

learning_rate: is the contribution of each model to the weights and defaults to 1

Parameters tuned by grid search:

- n_estimators=80 (The maximum number of estimators at which boosting is terminated). In case of perfect fit, the learning procedure is stopped early.
- learning rate =1

Score of optimized ada boost classifier is as follows

```
Optimized Model
------
Final accuracy score on the testing data: 0.6622
Final F-score on the testing data: 0.3850
```

IV. Results

Model Evaluation and Validation

Our optimized model received F-score 0.3850 on validation data and it seems to be good score as highest score in Kaggle leader board is approximate 0.44.

Model parameters values and description are as follows:

- n_estimators: The maximum number of estimators at which boosting is terminated). In case of perfect fit, the learning procedure is stopped early. In other words the number of models to iteratively train. In our case value is 80.
- learning_rate: is the contribution of each model to the weights and defaults to 1.
- base_estimator : (default=None)The base estimator from which the boosted ensemble is built. If None, then the base estimator is DecisionTreeClassifier(max_depth=1)
- algorithm : {'SAMME', 'SAMME.R'}, optional (default='SAMME.R') If 'SAMME.R' then use the SAMME.R real boosting algorithm. `base_estimator` must support calculation of class probabilities. If 'SAMME' then use the SAMME discrete boosting

algorithm. The SAMME.R algorithm typically converges faster than SAMME, achieving a lower test error with fewer boosting iterations.

- random_state : int, RandomState instance or None, optional (default=None):If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.(I have given it to 2 to get similar results after each execution.

I have obtained 10-fold cross validation score of Adaboost and random forest models. Scores are as follows:

Cross Validation	Model	Mean F1 score	std
10-fold	Adaboost	0.3131	0.0692
10-fold	Random Forest	0.3623	0.0471

Validating results in this way always help in assessing model robustness and how it will perform in future data. In our case Adaboost model has mean-score 0.3131 with standard deviation 0.0692 and random forest has mean cv score 0.3623 with standard deviation of 0.0471.

Let's see results of each fold for both models:

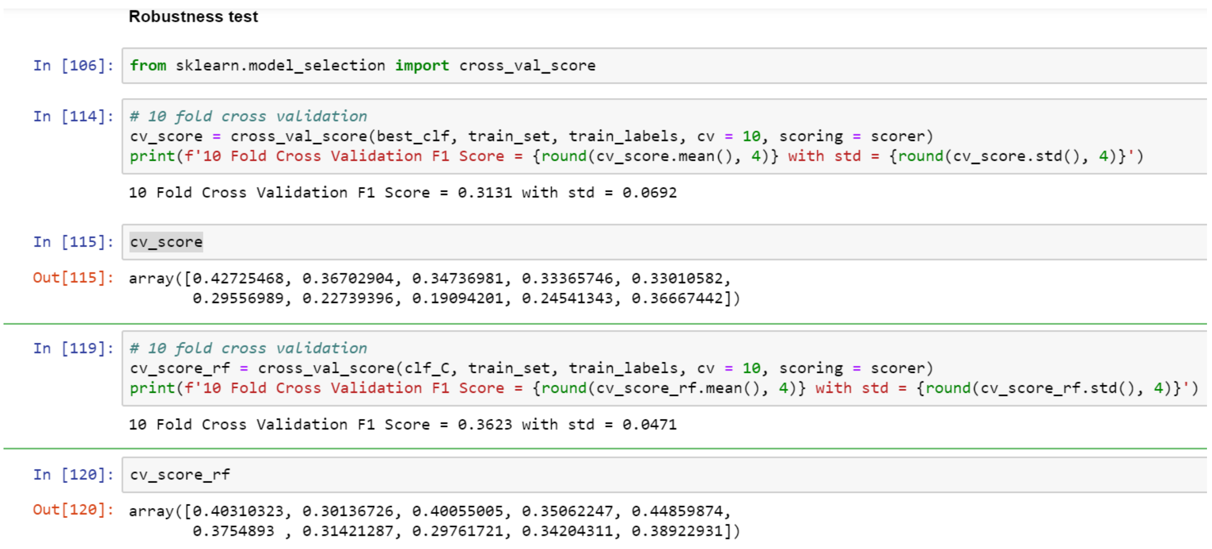
Adaboost classifier:

```
array([0.42725468, 0.36702904, 0.34736981, 0.33365746, 0.33010582,
       0.29556989, 0.22739396, 0.19094201, 0.24541343, 0.36667442])
```

Random Forest Classifer:

```
array([0.40310323, 0.30136726, 0.40055005, 0.35062247, 0.44859874,
       0.3754893 , 0.31421287, 0.29761721, 0.34204311, 0.38922931])
```

Above observations reflects that random forest classifier is outperforming Adaboost classifier. Adaboost has high standard deviation in F-score across multiple folds which means model is bit sensitive to data distribution. We might need more hyper parameter tuning and feature-engineering to nudge its performance.



Robustness test

```
In [106]: from sklearn.model_selection import cross_val_score

In [114]: # 10 fold cross validation
          cv_score = cross_val_score(best_clf, train_set, train_labels, cv = 10, scoring = scorer)
          print(f'10 Fold Cross Validation F1 Score = {round(cv_score.mean(), 4)} with std = {round(cv_score.std(), 4)}')

          10 Fold Cross Validation F1 Score = 0.3131 with std = 0.0692

In [115]: cv_score

Out[115]: array([0.42725468, 0.36702904, 0.34736981, 0.33365746, 0.33010582,
                 0.29556989, 0.22739396, 0.19094201, 0.24541343, 0.36667442])

In [119]: # 10 fold cross validation
          cv_score_rf = cross_val_score(clf_C, train_set, train_labels, cv = 10, scoring = scorer)
          print(f'10 Fold Cross Validation F1 Score = {round(cv_score_rf.mean(), 4)} with std = {round(cv_score_rf.std(), 4)}')

          10 Fold Cross Validation F1 Score = 0.3623 with std = 0.0471

In [120]: cv_score_rf

Out[120]: array([0.40310323, 0.30136726, 0.40055005, 0.35062247, 0.44859874,
                 0.3754893 , 0.31421287, 0.29761721, 0.34204311, 0.38922931])
```

Justification

I have followed all steps: data exploration, visualization, pre-processing, baseline model, parameter tuning etc while solving this problem as we should do in typical machine learning problem and achieved F-score 0.3850.
Base model score was 0.36 and then I have applied grid search to find better set of parameters and it helped increase score to 0.3850.

V. Conclusion

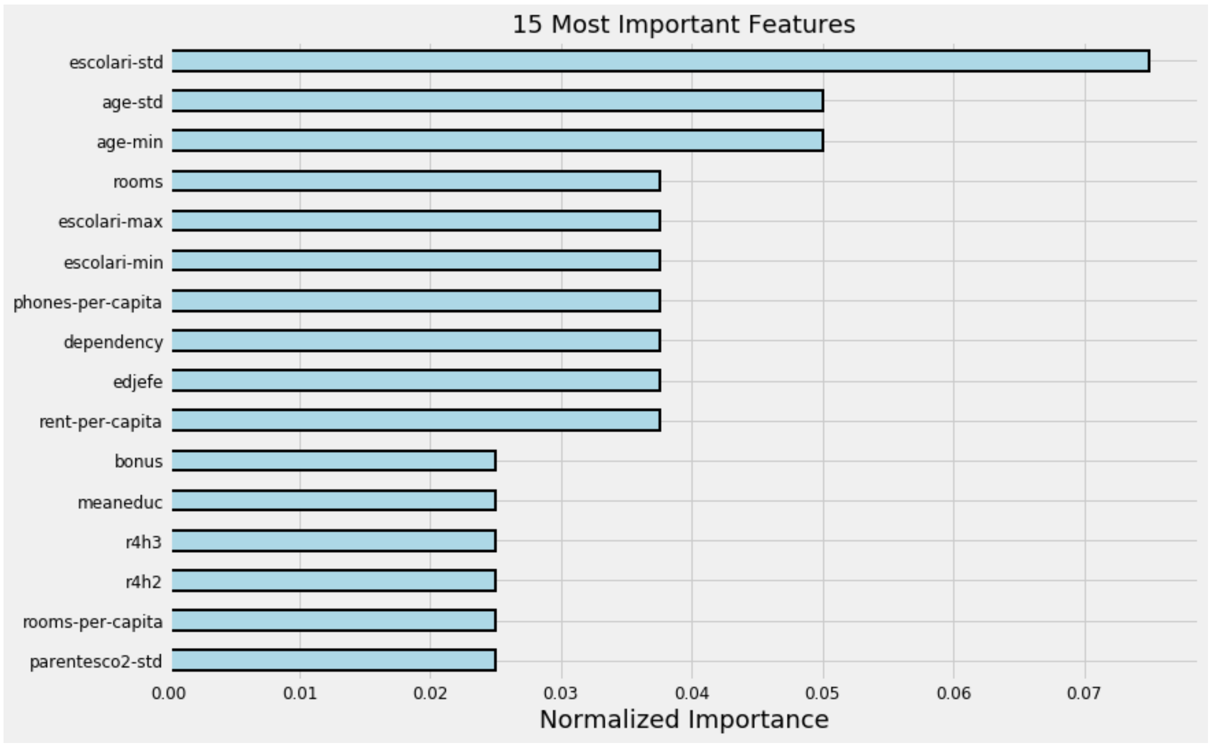
Free-Form Visualization

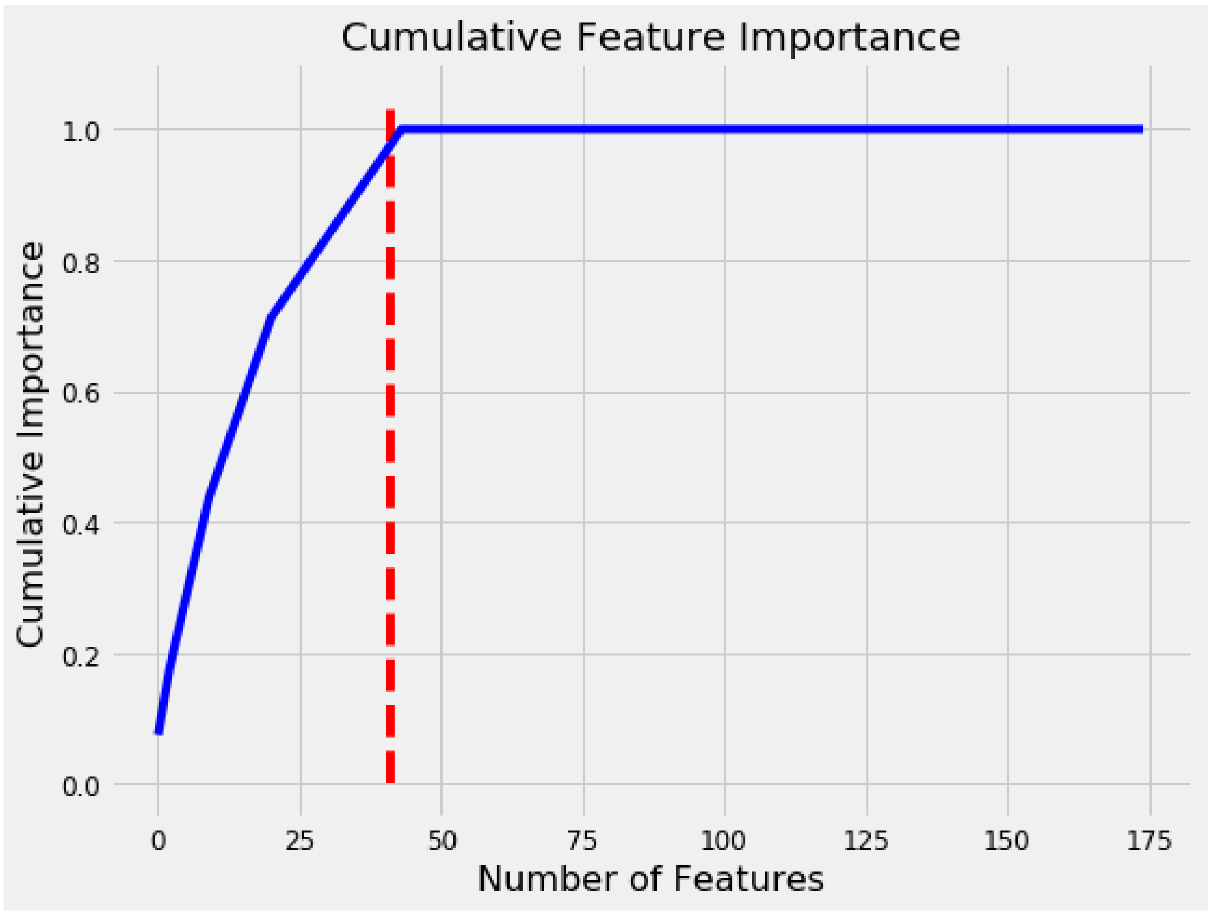
I would like to specify the importance of feature selection. Especially it becomes more important when feature space is high. We must try different set of combinations of features based on their importance to create better models.

Here I Have drawn two plots specific to our features.First plot dispalys 15 important features in order of their weights.

Second plot show very important observation: 41 features required for 95% of cumulative importance . This redirects us to take two actions: 1) Remove unimportant features and/or 2) Do more feature eng ineering and we can see how important newly created features would be.

This iterative approach will really help us to get robust model even out of very complicated data set wi th huge number of features.





Reflection

I started solving this problem with data exploration and visualization (EDA: exploratory data analysis).EDA plays very important role which solving any machine learning problem.
It helped me to gain deep understanding of features and its relationship with target variable.
As in most real word problems data doesn't come handy and it requires a lot of efforts to clean and engineer it before making it ready for modelling. I have followed below steps in pre-processing stage:

1) Converting few Boolean values into numbers in some features to make data type consistent in a feature
2) Correcting labels
3) Handling missing values
4) Feature engineering: new feature creation and elimination of redundant/correlated features.
5) Combining multiple Boolean features into ordinal variable
6) Aggregation of individual features into household features

Once our data is clean we can start our modelling task. As we have little class imbalance here, I used stratified cross validation to split our data into train and validation set.

After following above steps,I have trained a random forest model with default parameters to get a baseline F1 score on validation test. Then I created a train_predict pipeline so that I could quickly train couple of supervised learning algorithms and choose the best one for this dataset. After getting best model, I used grid search to find best set of hypermeters for chosen model and it helped me to nudge baseline F1 score

Interesting aspects of this project is “It relates to real word domain which really requires automated and accurate solution and help solving needs of poor in an effective manner”. Another aspect is great learning experience as feature set is large and bit complicated and it helped me gain deeper understanding of feature engineering,

Difficult aspect of this project is aggregating individual level features into household features so that we can effectively us provided data for training purpose.

Improvement

Though I think F1-score :0.3850 is okay, it can be much better if we further improve as:

1) Try other popular boosting methods as light GBM, XG boost or neural networks for structured data
2) We should do more feature engineering to identify important features and eliminate redundant features
3) As we have little class imbalance, we can consider oversampling techniques to handle this.
4) We can improve data visualization.