Costa Rica Household Poverty Level Prediction

Vivek Adrakatti
Madhav Mittal
Aryaman Shaan
Alex Lim

Introduction

This competition was created to make a system that can classify people into different income levels. Based on their income level, those in need of aid and financial assistance can be reached and the aid can be given to those who need it the most.

The Competition was created by the Inter-American Development Bank to improve the existing methods such as PMT

01 Exploratory Data Analysis

Make Sense of Data and Visualize it

02,03 Data Pre-Processing & Feature Engineering

Optimizing Data for Model-Building

04 Models

Building Machine Learning Models to Predict Poverty Level

05 Conclusion

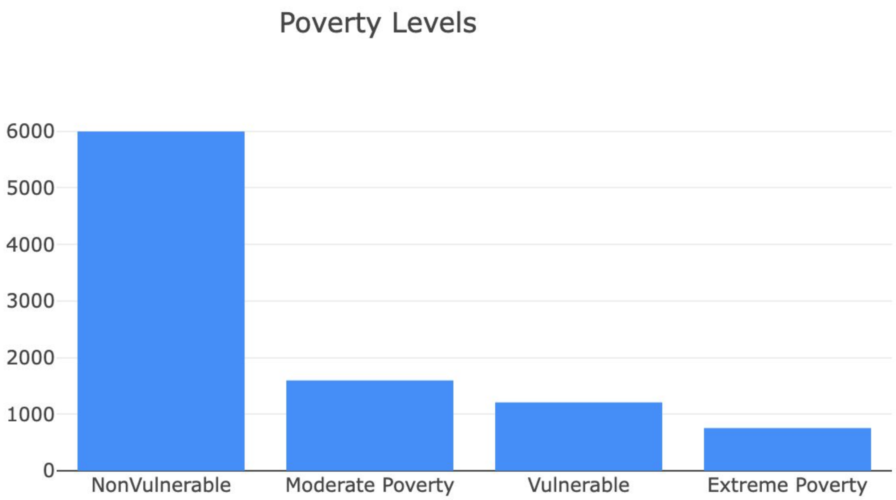
01
Exploratory Data Analysis

Missing Values in Dataset

	Feature	Number of NaN	Description
0	rez_esc	7928	Years behind in school
1	v18q1	7342	number of tablets household owns
2	v2a1	6860	Monthly rent payment
3	meaneduc	5	average years of education for adults (18+)
4	SQBmeaned	5	square of the mean years of education of adult...

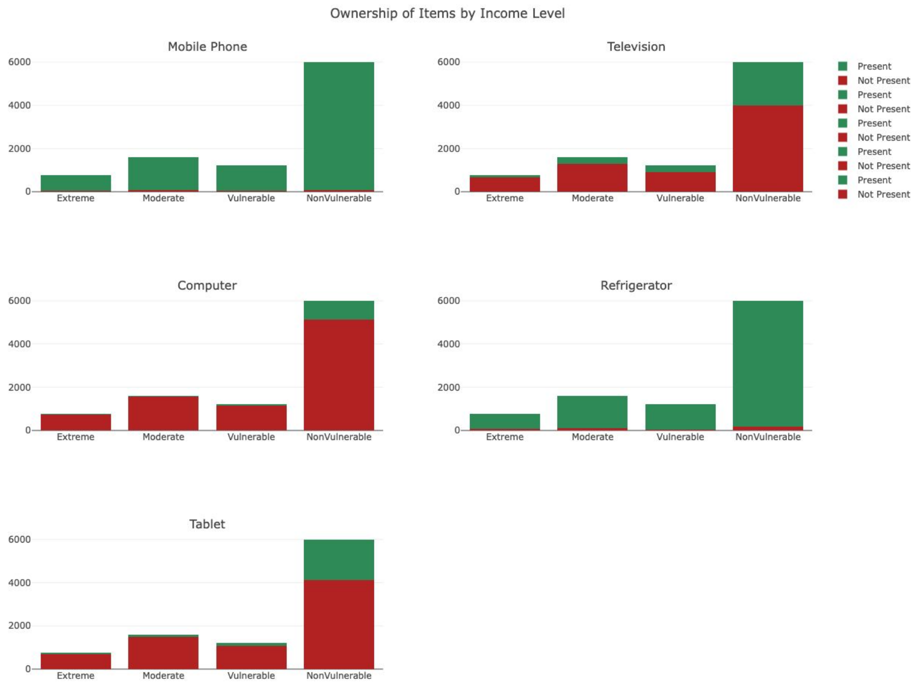
- NaN values affect model building as well as inferences that can be made from data
- Must be imputed during data preprocessing

Distribution of Population by Income Level

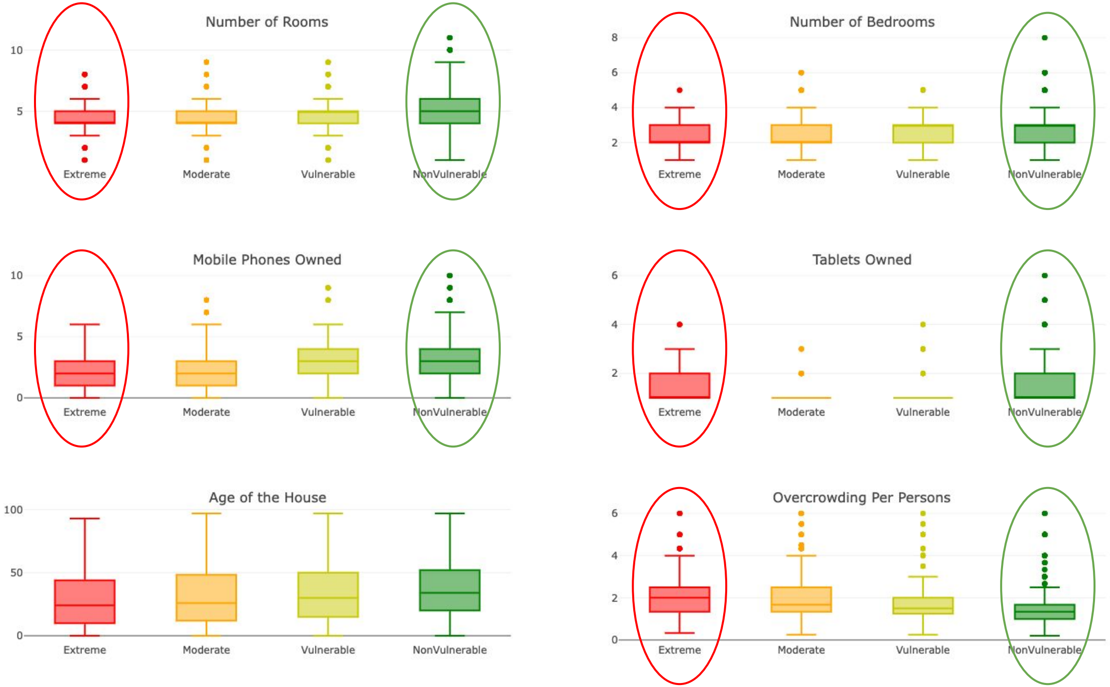


- Non Uniform Class Distribution
- Number of People in the Non-Vulnerable Category more than three times other categories
- This may affect model performance
- To visualize other metrics, we normalise the number of people from each Poverty Level

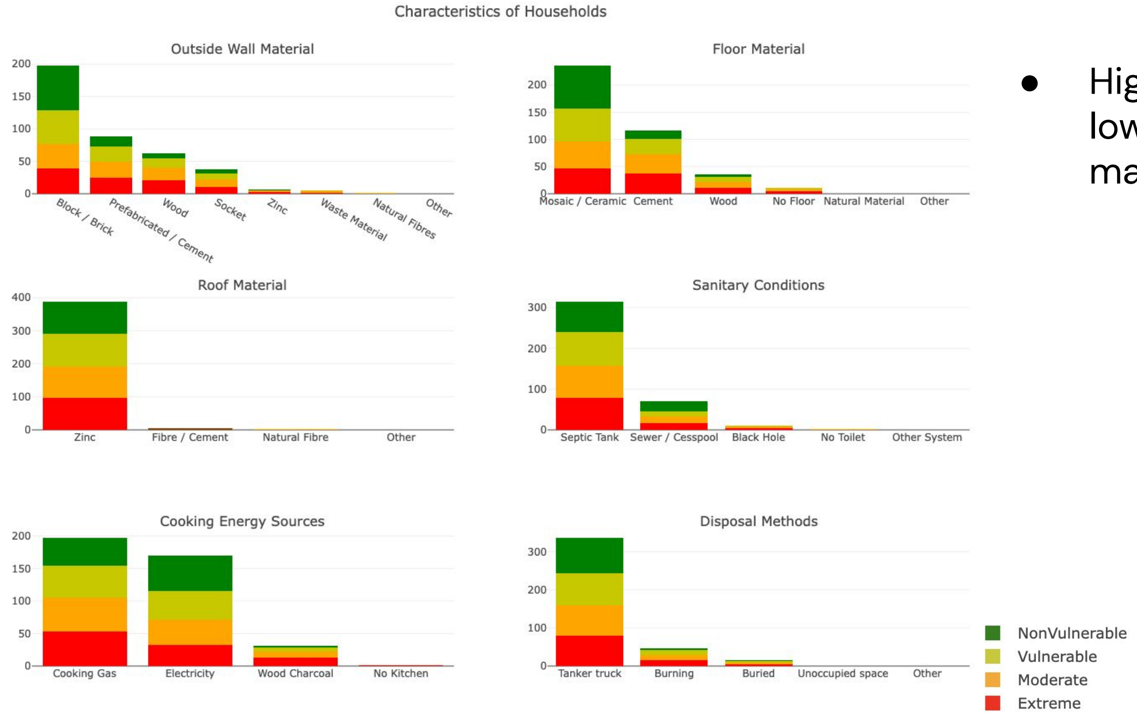
Graph Showing Ownership by Income Level



Graph Showing Ownership by Income Level

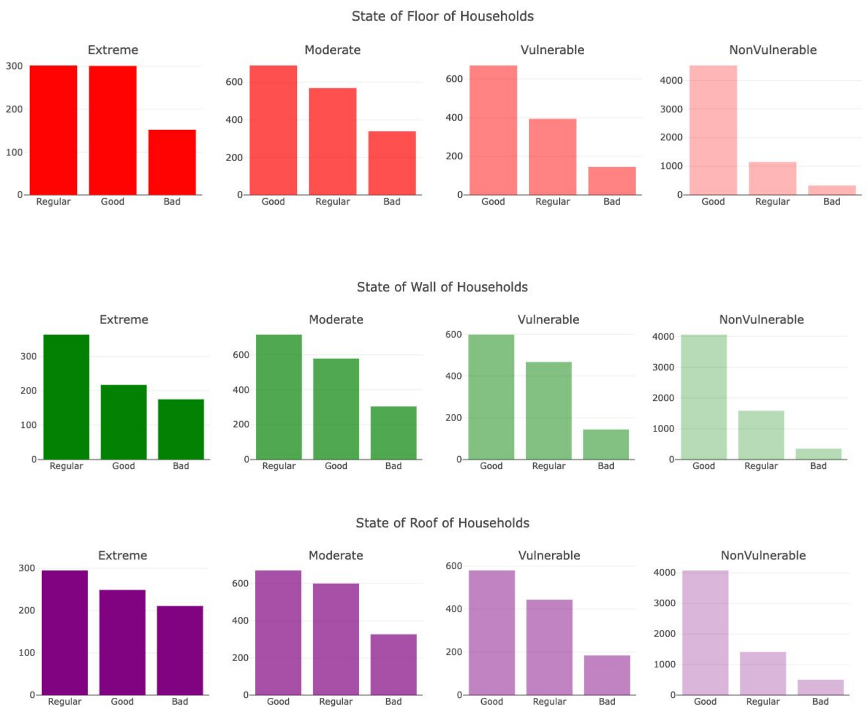


Household Materials and Characteristics



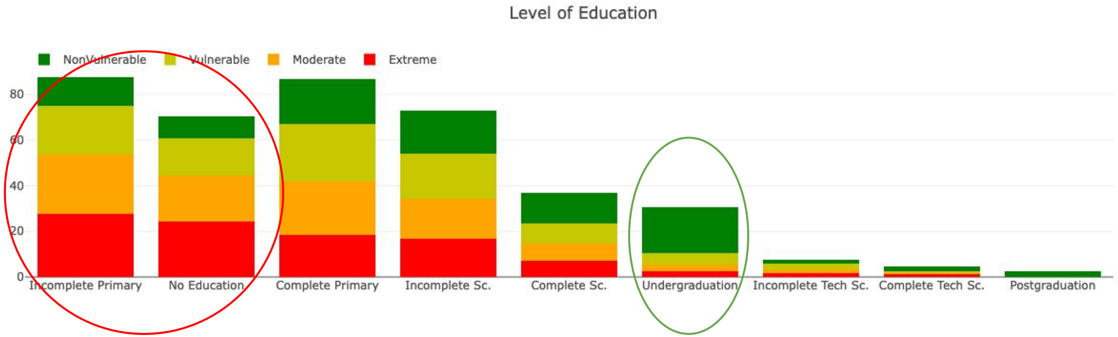
- Higher percentage of poorer people with lower grade characteristics and household materials

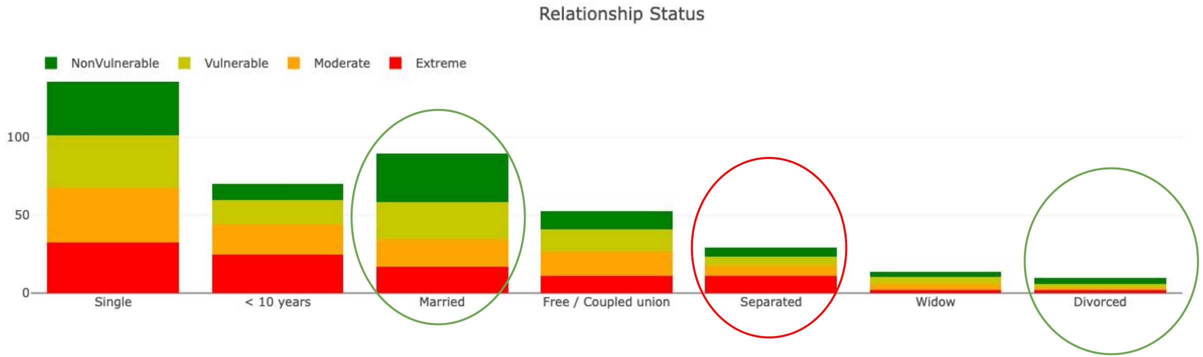
State of Houses according to Income Group



- Richer income groups have most houses with floors,walls and roofs in ‘Good’ condition
- Poorer income groups have most houses with floors, wall and roofs in ‘Regular’ condition

Distribution of Various Features by Income Levels





02
Pre - processing

Pre - processing steps

Converting Data-types

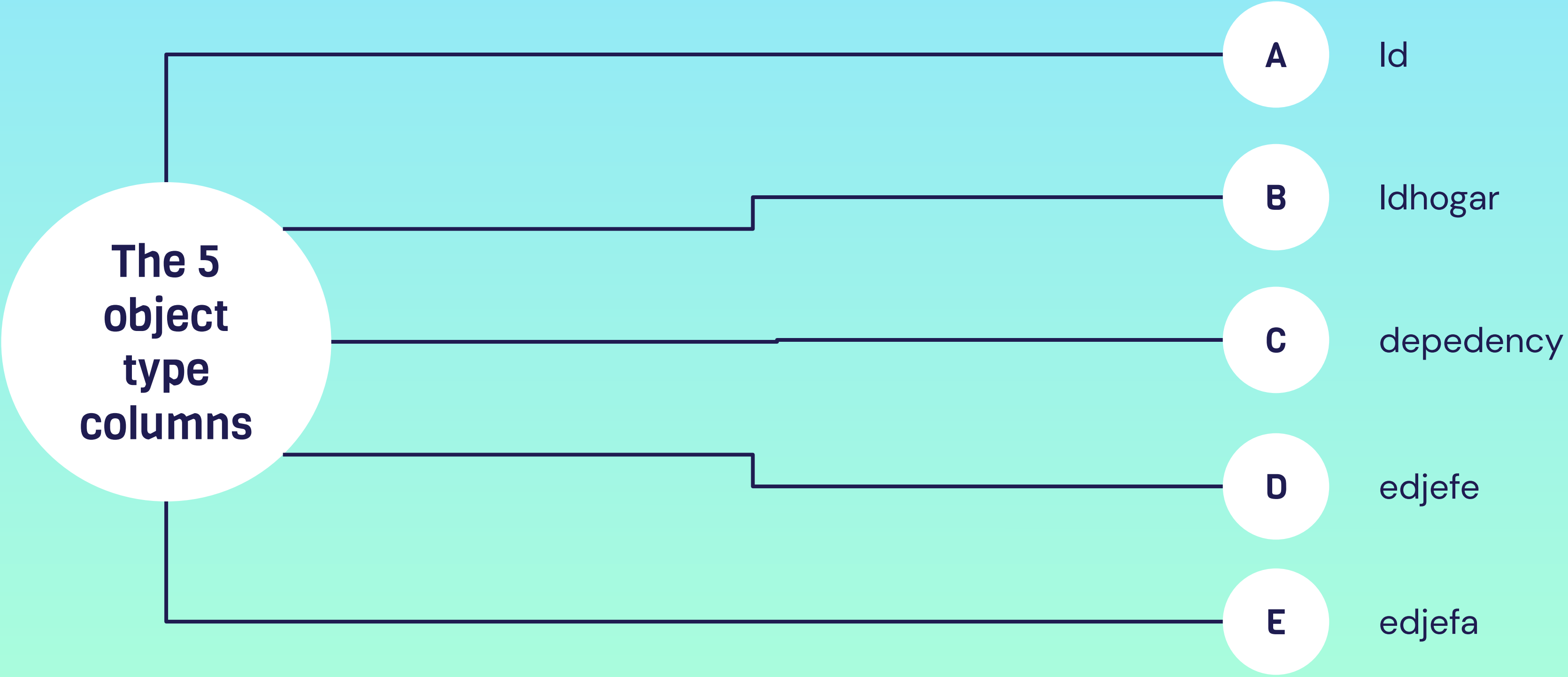
Identifying Labelling Errors

Filling Missing Values

1. Converting *Object* datatypes to *int/ float*

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23856 entries, 0 to 23855
Columns: 142 entries, Id to agesq
dtypes: float64(8), int64(129), object(5)
memory usage: 25.8+ MB
```



The 5 object type columns
A
Id
B
Idhogar
C
depedency
D
edjefe
E
edjefa

dependency, edjefe and *edjefa.*

- mix of floats and strings
- map "no" to 0, and "yes" to 1

```
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)
```

2. Identifying Labelling Errors

- Not all households have the same labels for all of their family members.
- It was seen that there were 85 such households.
- Easily fixed by re-labelling all the family poverty levels the same as the family head's poverty level.

```
# Figure out the number of unique values
all_equal = train.groupby('idhogar')['Target']
           .apply(lambda x: x.nunique() == 1)

# Households where targets are not all equal
not_equal = all_equal[all_equal != True]
```

- Some of the households had no head of the family
- We need to drop such rows.
- It was found that there were 15 such households in the training dataset, and 18 in the test dataset, all of which had to be dropped.

```
all_heads= train.groupby('idhogar')
              ['parentesco1'].sum()
no_heads= train.loc[train['idhogar'].isin
    (all_heads[all_heads == 0].index), :]
```

3. Filling Missing Values

```python
# Number of missing in each column
missing = pd.DataFrame(data.isnull()
          .sum()).rename(columns = {0: 'total'})

# Create a percentage missing
missing['percent'] = missing['total'] / len(data)

missing.sort_values('percent', ascending = False)
       .head(10).drop('Target')
```

We can first take a look at the percentage of values missing in each column

We find out -

- *Rez_esc* - 82.54% values are missing
- *V18q1* - 76.22% values are missing
- *V2a1* - 72.61% values are missing
- *SQBmeaned* - 0.10% values are missing
- *Meaneduc* - 0.01% values are missing

rez_esc: years behind in school

- Null value => no children currently in school.
- Variable only defined for individuals between 7 and 19.
- Anyone out of this range set to 0.

```
data.loc[((data['age']>19) |
         (data['age']<7)) &
         (data['rez_esc'].isnull()),
         'rez_esc'] = 0
```

v18q1: number of tablets owned by the family

- Null value => Family owns no tablets
- Fill the *nan* rows with a "0".

```
data['v18q1'] = data['v18q1'].fillna(0)
```

v2a1: monthly rent payment

- Null value => households that own a home and thus don't pay any rent
- Fill the *nan* rows with a "0"

```
data['v2a1'] = data['v2a1'].fillna(0)
```

03
Feature Engineering

Feature Engineering Steps

Dropping unnecessary features

Creating new features by dropping old ones

Converting individual features into household features through aggregation

1. Dropping Unnecessary Features

Dropping of squared columns

```
squared_columns = [x for x in [*train_full] if 'sq' in x or 'SQ' in x]
data = data.drop(columns = squared_columns)
```

Dropping of highly correlated columns

- Columns that have correlation greater than 95% - ***tamhog', 'hhsize', 'coopele', 'hogar_total', 'area2', 'Female'***
- Perfectly correlated features => dropped.
- If not => create new, more useful features.

The following have perfect correlations. The former of the pairs/ triplets are thus dropped

1. *Tamhog, hogar_total* and *hhsize*
2. *R4t3* and *hhsize*
3. *Area1* and *area2*
4. *Male* and *female*

```
squared_columns = [x for x in [*train_full] if 'sq' in x or 'SQ' in x]
data = data.drop(columns = squared_columns)
```

hhsize and tamviv

- *hhsize* and *tamviv* => not perfectly correlated.
- Reason => some family members might not be living in the household.
- New Feature => tamviv - *hhsize*

```
heads['hhsize-diff'] = heads['tamviv'] - heads['hhsize']
```

Coopele

- *coopele* and *public* => *not* perfectly correlated.
- Reason => coopele and public are not exhaustive
- New Feature => *elec, which* can take on four values - *noelec* (0), *coopele* (1), *public* (2) and *planpri* (3).

2. Creating new features by merging old ones

Merging *epared1,2,3*, *etecho1,2,3* and *eviv1,2,3* columns

- *walls/roof/floor* => three binary columns each (*bad, average or good)*
- Merge into one, ordinal column each

```
heads['walls'] = np.argmax(np.array(heads[['epared1', 'epared2', 'epared3']]), axis = 1)
heads['roof'] = np.argmax(np.array(heads[['etecho1', 'etecho2', 'etecho3']]), axis = 1)
heads['floor'] = np.argmax(np.array(heads[['eviv1', 'eviv2', 'eviv3']]), axis = 1)

heads = heads.drop(columns = ['epared1', 'epared2', 'epared3', 'etecho1', 'etecho2',
  'etecho3', 'eviv1', 'eviv2', 'eviv3'])
```

```
heads['walls+roof+floor'] = heads['walls'] + heads['roof'] + heads['floor']
```

Walls, *floor* and *roof* can be further merged to create a new variable, which might be a good overall feature

Merging *sanitario, elec, pisonotiene, abastaguano* and *cielrazo* columns

- Variables that tell us the relative condition of the house
- Merge to understand the overall quality of the house
- Might create an interesting feature

```
heads['warning'] = 1 * (heads['sanitario1'] + (heads['elec'] == 0) +
                        heads['pisonotiene'] + heads['abastaguano'] +
                        (heads['cielorazo'] == 0))
```

Merging *refrig, computer, v18q1* and *television* columns

- Amenities that not all houses have
- Merging may help us distinguish privileged (or rich) households from the others.

```
heads['bonus'] = 1 * (heads['refrig'] + heads['computer'] +
                      (heads['v18q1'] > 0) + heads['television'])
```

Creating new features using intuitive measurements

```
heads['phones-per-capita'] = heads['qmobilephone'] / heads['tamviv']
heads['tablets-per-capita'] = heads['v18q1'] / heads['tamviv']
heads['rooms-per-capita'] = heads['rooms'] / heads['tamviv']
heads['rent-per-capita'] = heads['v2a1'] / heads['tamviv']

ind['escolari/age'] = ind['escolari'] / ind['age']
ind['inst/age'] = ind['inst'] / ind['age']
ind['tech'] = ind['v18q'] + ind['mobilephone']
```

3. Converting individual features into household features through aggregations

- Convert all the features that are on an individual level to household level.
- Six aggregations, namely - *min, max, sum, count, std* and *range.*
- **These aggregations will be applied to each household by grouping on *idhogar.***

```
# Group and aggregate
ind_agg = ind.drop(columns = 'Target').groupby('idhogar')
.agg(['min', 'max', 'sum', 'count', 'std', 'range'])
```

- We obtain six times the number of attributes as before.
- A lot of them are useless since they portray similar information and thus have a high correlation.
- **We drop all aggregated columns with correlation greater than 0.95.**

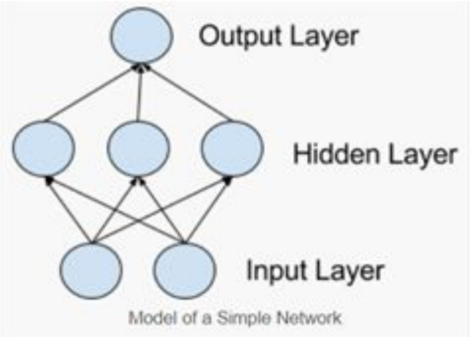
```
to_drop = [column for column in upper.columns
           if any(abs(upper[column]) > 0.95)]
```

04

Models

Approach 1: Multilayer perceptron (MLP)

MLP neuron network



Output of a neuron

$$y = \varphi(\sum_{i=1}^{n} w_i x_i + b) = \varphi(\mathbf{w}^T \mathbf{x} + b)$$

Approach 1: Multilayer perceptron (MLP)

Motivation

- Common baseline neural network model
- Easy to implement
- Suitable for classification
- Works well with tabular data

Experiment

- Activation: ReLu
- Solver: Adam

Approach 1: Multilayer perceptron (MLP)

Results

MLP costa
(version 4/8)
2 hours ago by alex

Notebook MLP costa | Version 4

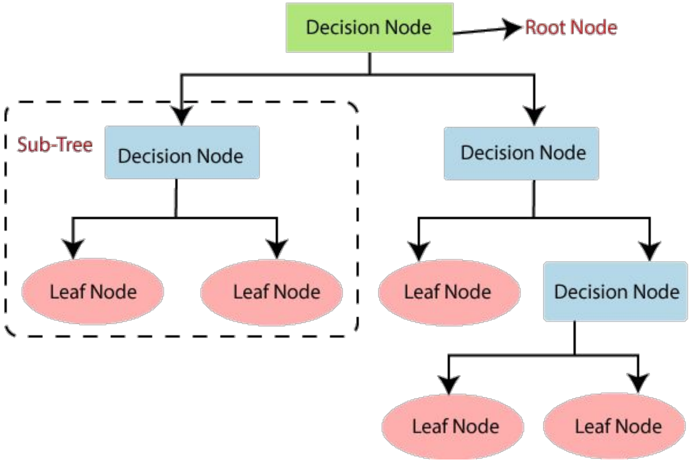
0.36389

Rank: 433

Approach 2: Decision Trees

- Root node is at the top of the tree
- At each node a decision is made to perform a split
- Recursive Splitting technique to choose which features to split on
- Metric: information gain or gini impurity

Splitting of Nodes



Approach 2: Decision Trees

Motivation

- Simple to understand, visualize and interpret.
- Can handle numerical and categorical data
- Internal Feature Selection
- Baseline model

Experiment

- Criteria: gini
- Max features for split: sqrt * no of features
- Max depth: None
- Take mean of the results as the predictions from k-fold cross validation

Approach 2: Decision Trees

Results

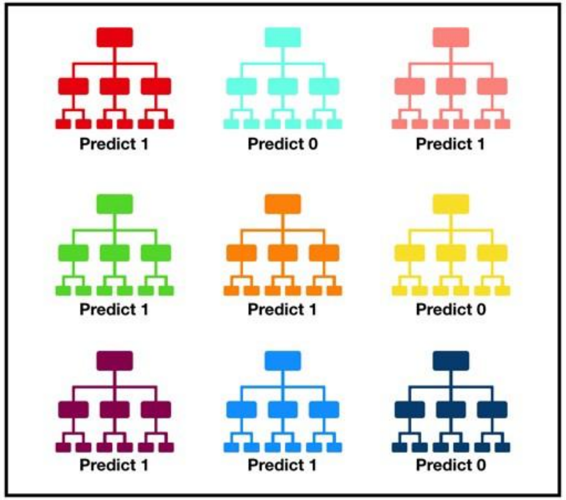
Decision_trees1
(version 1/1)
just now by aryaman shaan
Notebook Decision_trees1 | Version 1

0.34906

Rank: 474

Approach 3: Random Forest

- Ensemble based on Decision trees
- Bootstrapping features for each tree
- Each tree has different strength and weaknesses
- Final Voting



Tally: Six 1s and Three 0s
Prediction: 1

Approach 3: Random Forest

Motivation

- One Decision tree can Overfit Data
- Multiple trees mitigate each others’ mistakes
- Can better find global optimum
- Works well with tabular data

Experiment

- No of estimator: 100
- criteria: gini
- Max features: sqrt
- Max depth: None
- K Fold cross validation
- Take mean of the results as the predictions

Approach 3: Random Forest

Results

Random Forest Classifier 2 (version 3/3) 0.37976

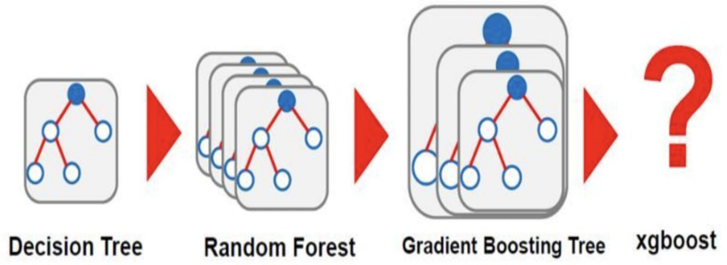
just now by aryaman shaan

Notebook Random Forest Classifier 2 | Version 3

Rank: 363

Approach 4: XGBoost

- Tree based ensemble learning algorithm
- Gradient Boosting Framework: Trees made iteratively learning from previous trees errors using gradient descent.
- Quality Over Quantity: Each tree is not split to full extent



Approach 4: XGBoost

max_depth	n_estimators	Mean F1	Std deviation F1
5	10	0.276	0.035
5	20	0.269	0.035
5	30	0.253	0.034
5	40	0.233	0.025
5	50	0.214	0.029
10	10	0.269	0.034
10	20	0.245	0.038
10	30	0.252	0.039
10	40	0.242	0.038
10	50	0.231	0.026

Motivation

- Good Results for tabular Data
- Has in-built (L1 & L2) regularisation
- Fast: Cache aware
- Tree pruning start after maximum depth in a branch is reached

Experiment

- Varied max_depth and n_estimators
- Selected Model by Stratified K-fold validation
- Trained the optimum model found on multiple k-folds
- Took mean of predictions of the models on each fold

Approach 4: XGBoost

Results

xgboost3
version 4 (version 3/3)
just now by aryaman shaan
Notebook xgboost3 | version 4

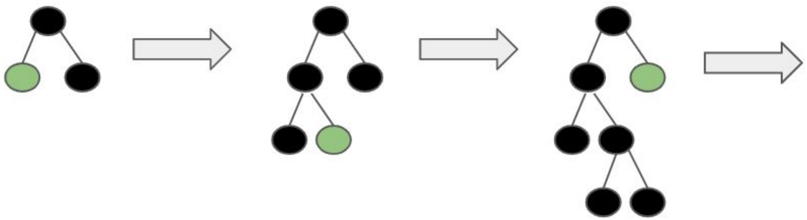
0.37048

Rank: 410

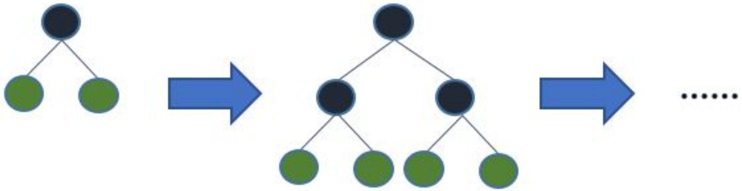
Approach 5: LightGBM

LightGBM

LightGBM leaf-wise



XGBoost



Level-wise tree growth

Approach 5: LightGBM

Motivation

- Fast (40 Seconds training time)
- High accuracy
- Suitable for tabular and small dataset

Experiment

- Bayesian optimisation for hyperparameter tuning
- Stratified 7-fold cross validation
- Take mean of the results as the predictions

Approach 5: LightGBM

Results

Costa LightGBM
(version 1/)
just now by alex

0.44202

Notebook Costa LightGBM | Version 1

Rank: 26

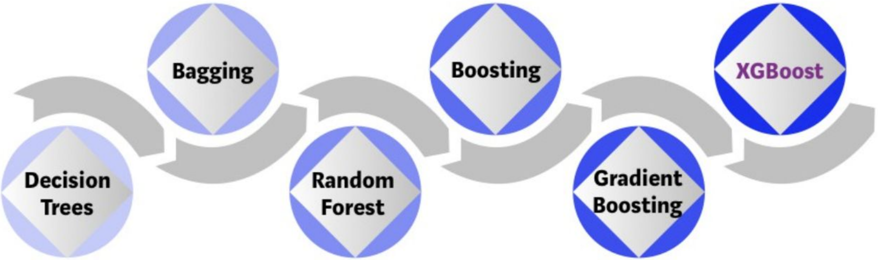
05

Conclusion

OBSERVATIONS - MODELS

- MLP: Insufficient Training Data
- Random Forest: Better results
- XGBoost: Overfitting
- **LGBM: best results from tree based ensemble**

CONCLUSION



- Exploratory Data Analysis Helps
- Overfitting on train data reduces performance of models
- The Combination of Feature Engineering and Model Decides Performance

Thank you!