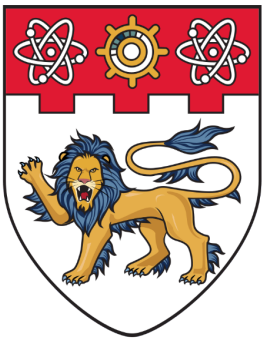


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4041 - MACHINE LEARNING

COURSE PROJECT REPORT

Costa Rican Household Poverty Level Prediction

Group 8

Contribution

Akshat Sharma	SVM, LightGBM, Hyperparameter Tuning, Voting Classifier, Exploratory Data Analysis, Feature Engineering
Garg Astha	Exploratory Data Analysis, Feature Engineering, Decision Tree Classifier, Random Forest Classifier
Musthiri Sajna	Exploratory Data Analysis, Feature Engineering
Ramasubramanian Nisha	Exploratory Data Analysis, Decision Tree Classifier, Random Forest Classifier, Adaptive Boosting, Gradient Boosting, Stacking, Hyperparameter Tuning

Table of Contents

1. Problem Statement

The problem statement we chose “Costa Rican Household Poverty Level Prediction” is for classifying households that are in dire need of social welfare assistance.The motivation for solving this problem using Data Science techniques lies in the fact that the poorest cannot provide required income and expense records to prove that they qualify for financial aid.

The traditional econometrics approach is to use Proxy Means Test (PMT), most commonly used in Latin America. This method uses the family’s observable attributes to categorise and predict their level of need. However, the accuracy of the PMT method remains a big issue as the population grows and poverty declines.

Therefore, Inter-American Development Bank has provided the data to the Kaggle Community to devise novel solutions using Machine Learning methods to identify these households with improved accuracy. These solutions can be used to accurately gauge the level of social need in other countries as well.

2. Evaluation on Kaggle

The submissions will be evaluated based on their macro F1 score on the test dataset. F1 score is defined as the harmonic mean of precision and recall. It can be calculated as follows:

$$\text{F1 Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

In terms of true positives, false positives and false negatives, it would be as follows:

$$\text{F1 Score} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

The alternative equation for F1 score | Image by author

This is a Kernels-Only Competition, so we had to make submissions directly from Kaggle Kernels. According to the rules of the competition, only heads of household will be scored. The macro F1 score is computed as the arithmetic mean of all per-class F1 scores.

3. Description of Dataset

The data folder consists of train.csv and test.csv with 9557 rows and 23856 rows respectively. The column “target” determines the poverty level of a household. The test.csv does not contain this column. Hence, the train.set is used as the entire data set for the model preparation. The train.scv contains 143 columns and each record is associated with a single person. The description for each column is given in the kaggle notebook.

4. Data Preprocessing

The columns of the original data set have been changed to shorter English descriptions for better understanding. Hence, all the data will be referenced using their renamed columns.

4.1 Data Exploration

The poverty levels of the households are very unbalanced which is obtained from the target size of the train.csv as shown in Figure 1. The number rows belonging to class 'four' form 63% of the data set but the number of rows belonging to class 'one' account to only 8%.

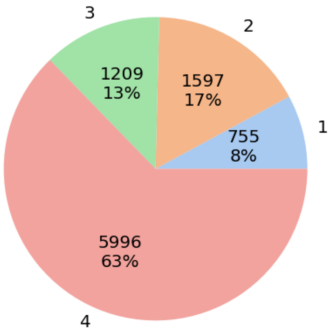


Figure 1

4.1.1. Handling missing values

There are 6860 rows with "rent" that have NaN values. The missing values are also found in 7342 rows of "no_of_tablets" and 7928 rows of "years_behind school". This uneven distribution in household sets can be seen in figures 2 and 3.

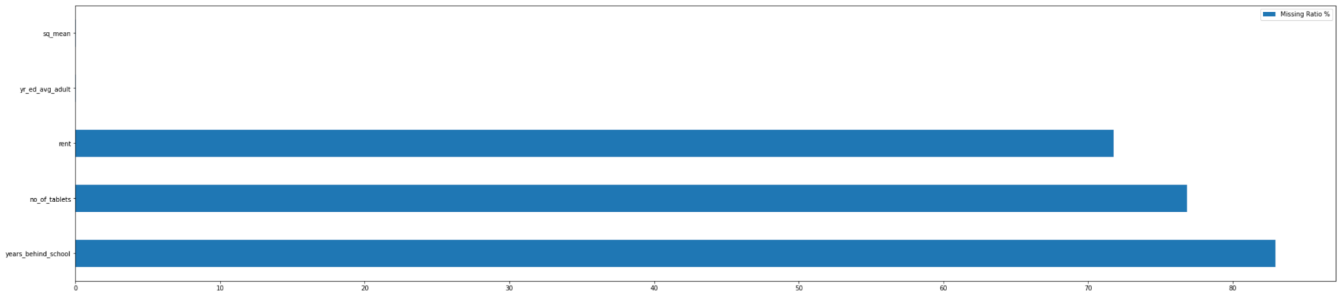


Figure 2

	total	Percent
years_behind_school	7928	0.829549
no_of_tablets	7342	0.768233
rent	6860	0.717798
sq_mean	5	0.000523
yr_ed_avg_adult	5	0.000523
Id	0	0.000000
no_of_adults	0	0.000000
sibling_inlaw	0	0.000000
other_member	0	0.000000
other_non_member	0	0.000000

Figure 3

The value in column “no_of_tablets” describes if the household has a tablet or not. If the value is Nan then, we can assume the value to be 0. Hence, the missing values in the columns: “years_behind school”, “no_of_tablets” and “rent” have been replaced with zeros. The boolean variables: yes and no for the “dependency” column have been replaced with 1 (yes) and 0 (no).

4.1.2. Checking for data inconsistency

The dependency rate for the household sets are as shown in figure 4.

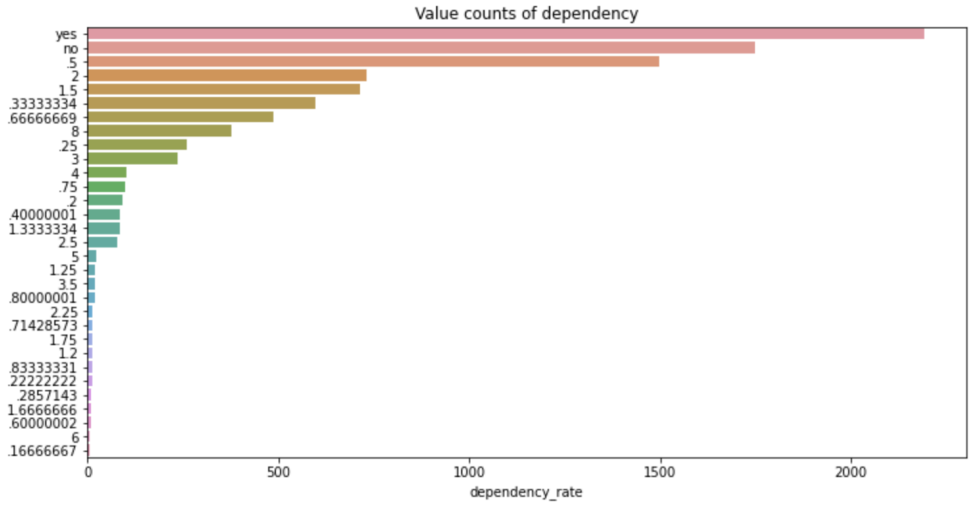


Figure 4

There are inconsistencies between the values under the “years_behind_school” and “age”. For example, the people with age less than 7 or greater than 19 cannot be in school practically and these errors cause the data to be inconsistent. This inconsistency can be noted in the figure 5 that displays the frequency of non null years in the school with respect to age.

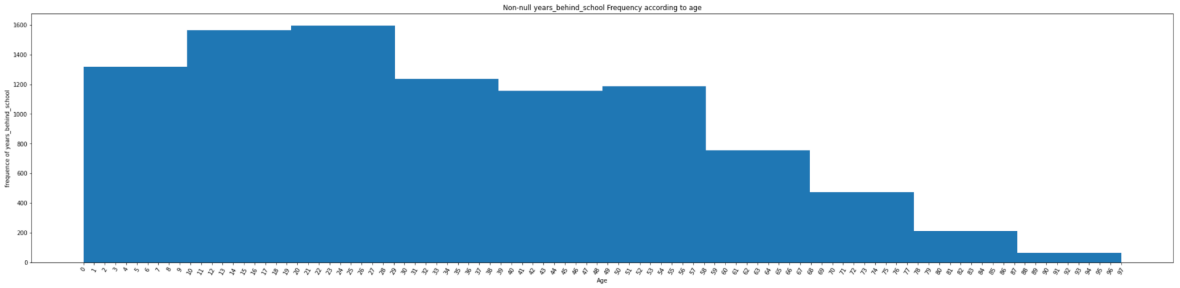


Figure 5

There are other inconsistencies such as some rows indicating that there are toilets and no toilets in the same household. For this inconsistency, if there is no water available to the household which depends on the value of “water_no”, the household is considered to have no toilet and the column “has_toilet” is set to 0.

Another notable inconsistency is the households with contradicting targets which is shown in figure 6.

	household_level	house_head	Target
282	4b6077882	1	1
283	4b6077882	0	2
284	4b6077882	0	2
285	6833ac5dc	0	2
286	6833ac5dc	0	2
...	...	...	...
9535	9bbf7c6ca	1	1
9536	9bbf7c6ca	0	2
9537	9bbf7c6ca	0	2
9538	9bbf7c6ca	0	2
9539	9bbf7c6ca	0	2

335 rows × 3 columns

Figure 6

This inconsistency is solved by grouping the household and target and re-assigning target to the households.

```
[ ]  for HH in HHID_Discrepancy:
         Targets= (train.loc[train['household_level']==HH,'Target'])

         if Targets.mode().shape[0] >1:
             for i in Targets.index:
                 if train.loc[i,'house_head']==1:
                     HeadTarget= train.loc[i,"Target"]
             for i in Targets.index:
                 train.loc[i,'Target']=HeadTarget
         elif Targets.mode().shape[0]==1:
             for i in Targets.index:
                 TrueTarget=int(Targets.mode())
                 train.loc[i,'Target']=TrueTarget

     train_ensemble=train.copy()
```

Figure 7

4.1.3. Dropping columns

The dataset is given consists of multiple duplicated columns. For example, columns such as “size_household” and “household_size” both refer to the number of people in a household. Hence, one column can be removed and the other is retained. Additionally, there are many columns with squared values of the same column. For example, the column “sq_age” is the squared value of age, “age”. So, this column “sq_age” is removed. For the same reason,all the squared value columns, “sq_yr_schooling”,
“Sqb_age”, “sq_individual”, “sq_ed_male_head”, “sq_children”, “sq_oc”, “sq_dependency”, “sq_mean”, “sq_age” are removed. These columns are removed and provide no additional information.

5. Feature Engineering

The predictive models we use and the results produced are directly affected by the data that is fed into the model. The better the features we prepare and choose, the better results we will obtain from the model. The features should define the structure of the data itself.

The usage of good features can help us in reducing the complexity of the model being implemented making it faster to run, understand and maintain.

The process of feature engineering includes creation of new features to represent the data, transformation of existing data and lastly, selecting and extracting features known as variables. This is done keeping the objective of creating the most conducive features for the model in mind.

5.1. Feature Creation

This process involves creating new features using the existing ones by manipulating them. Since the scoring will be done only on the household heads, we need to create features at the household level.

1. We created new features by calculating ratios for the various columns with respect to the total number of individuals in the household, or the household size, or the number of rooms depending on the type of the feature. This enabled an easier comparison between the different data values.

2. A new scoring system for the level of education was created. There were multiple features representing the education level, so we combined them to create one feature. This feature ranges from 0 to 6, higher number meaning the individual received more education than the one with lower number.

3. To incorporate the value of the features from other members of the household to that of the household head, we calculated the mean, maximum, minimum, sum, and standard deviation for several features. These included characteristics like education, marital status, gender, age, family information etc. By adding this information to the household heads rows it gives more information about the other members besides the head giving a better overview of the household.

4. Next we created a feature to assess the living condition of the household. This was done by combining the features of the wall, floor and the utilities available.

5.2. Feature Selection

It is a very important step in the preprocessing of data. It involves eliminating those features that are irrelevant or those that decrease the accuracy of the model while selecting those that can improve the performance of the model.

Correlation between the features

It is a way to determine if there is a relationship between various attributes as well as with the target variable. In the process of finding correlation between two attributes, there can be three possible results - positive correlation meaning both the attributes show similar trend i.e. if one increases the other also increases, negative correlation meaning they have opposite trend i.e. if one increases then the other decreases, or lastly, no correlation meaning they are independent of each other and one does not affect the other.

We calculated the correlation of different features by using the .corr() function provided by pandas which gives the Pearson Standard Correlation Coefficient by default. This helps in finding linear relationships between continuous variables, ranging from -1.0 (perfect negative correlation) to 1.0 (perfect positive correlation).

$$\text{Correlation}\,(X,Y) = \frac{s_{xy}}{s_x \times s_y}$$

where s_{xy} is the covariance of X and Y, which is calculated as:

$$s_{x\,y} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})$$

and s_x and s_y are the standard deviation of X and Y, respectively.

Fig Pearson Standard Correlation Coefficient explanation [1]

The output showed that some features “waste_throw_river”, ”under_10”, ”house_head”, ”partner”, ”child”, ”step_child”, ”child_inlaw”, ”grandchild”, ”parent”, ”parent_inlaw”, ”sibling”, “sibling_inlaw”, ”other_member” and ”other_non_member” gave the correlation as NaN meaning the standard deviation for the values was 0. This revealed that there were some features with only 1 unique value and thus, could not affect the target variable. As a result, these columns were dropped.

Chi-squared Test of Independence

This statistical hypothesis test is used to determine if 2 categorical variables are likely to be related or not. Thus it indicates if there is any association between 2 categorical features.

$$\chi^2 = \sum_{i=1}^{R} \sum_{j=1}^{C} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}$$

where

o_{ij} is the observed cell count in the i^{th} row and j^{th} column of the table

e_{ij} is the expected cell count in the i^{th} row and j^{th} column of the table, computed as

$$e_{ij} = \frac{\text{row } i \text{ total} * \text{col } j \text{ total}}{\text{grand total}}$$

Fig Chi-Squared test Calculation[3]

The first step is to state the hypothesis - null (H_0 - the two variables are independent) and the alternative (H_1 - they are not independent). The result of the test gives us the p-value. If the value is lesser than the chi-square critical value (in this case 5% i.e. 0.05), we reject the null hypothesis.
Applying this teston the categorical features of the given dataset, we find 122 attributes that have p-value between 0.0 to 0.05.

6. Models

Model 1: Support Vector Machine (SVM)

Overview

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

Motivation

“Support Vector Machine” (SVM) is a supervised machine learning algorithm that can be used for both classification or regression problems. However, it is mostly used in classification problems and in fact had been the state-of-the art algorithm for

classification problems until a few years ago until deep learning and neural networks emerged as a better technique to separate non-linear classification challenges.

Experiment and Result

Since the SVM classifiers do not require much preprocessing so initially the model was trained on data after performing exploratory data analysis without any further preprocessing. The train dataset was created by splitting in a 70:30 ratio, the data point corresponding to the household heads. However, the accuracy achieved was only 67% with the f1-score macro-average being just 29%.

	precision	recall	f1-score	support
1	0.45	0.07	0.12	75
2	0.45	0.09	0.15	141
3	0.33	0.05	0.08	86
4	0.68	0.97	0.80	590
accuracy			0.67	892
macro avg	0.48	0.29	0.29	892
weighted avg	0.59	0.67	0.57	892

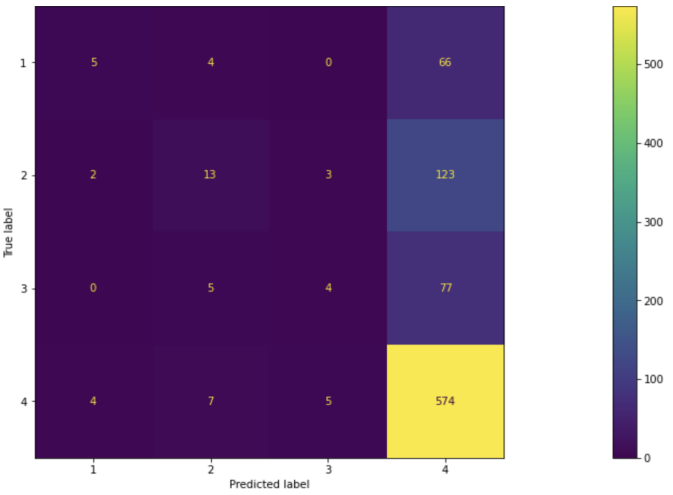


Fig: Classification Report and Confusion Matrix for SVM model

Later dimensionality reduction was done on the training dataset using PCA as the number of features being used were quite a lot but still there was minimal change in the accuracy and f1-score.

Conclusion

The SVM classifier did not work as expected with accuracy being around 67% both with and without PCA whereas the F1 score was around 0.30. A few potential reasons for this can be:

1. SVM algorithm is sensitive to outliers and fails to perform well when a high no. of outliers are present in the dataset which we believe is the case with our dataset.
2. Hyperparameters like cost (C) and gamma of SVM, is not that easy to fine-tune and also hard to visualise their impact
3. The SVM model is difficult to understand and interpret by human beings, unlike Decision Trees which makes it virtually impossible to narrow down the cause of failure of the SVM model for our dataset.

Model 2: Decision Tree Classifier

Overview

Decision trees as the name suggests create splits in the dataset by forming yes/no questions at every step until the data points belonging to each class are segregated. At every question a ‘node’ is added to the tree which branches based on the answer to the question.

Motivation

Decision Tree Classifiers are simple models to perform classification as they are easy to understand and interpret. It does not require heavy data preprocessing and the cost of using the tree itself is logarithmic in the number of data points used to train the tree. This way, they pose as the best initial models.

Experiment and Result

The decision tree classifiers do not require much preprocessing or preparation for the data. The train dataset was created by splitting in a 70:30 ratio, the data point corresponding to the household heads.
The results are as follows:

	precision	recall	f1-score	support
1	1.00	1.00	1.00	520
2	1.00	1.00	1.00	1135
3	1.00	1.00	1.00	864
4	1.00	1.00	1.00	4170
accuracy			1.00	6689
macro avg	1.00	1.00	1.00	6689
weighted avg	1.00	1.00	1.00	6689

	precision	recall	f1-score	support
1	0.15	0.21	0.18	63
2	0.26	0.24	0.25	136
3	0.19	0.19	0.19	107
4	0.76	0.75	0.76	586
accuracy			0.57	892
macro avg	0.34	0.35	0.34	892
weighted avg	0.57	0.57	0.57	892

Fig: Classification report for DTC training split and validation split

The confusion matrix for the validation split is as follows:

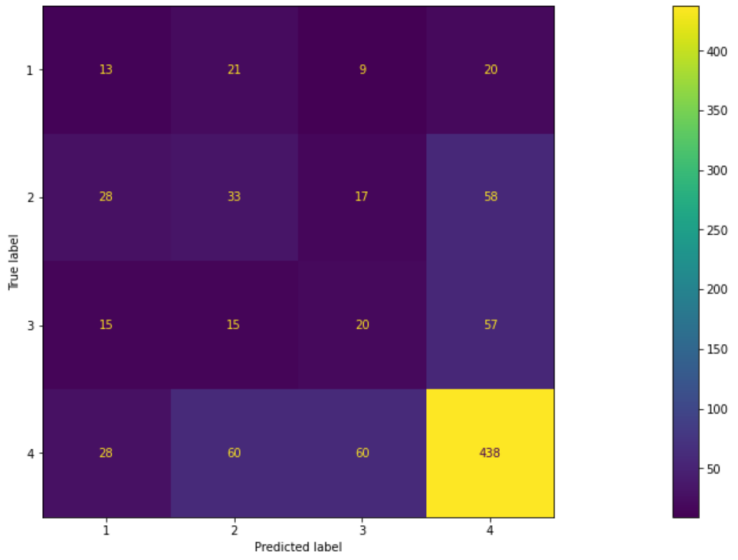
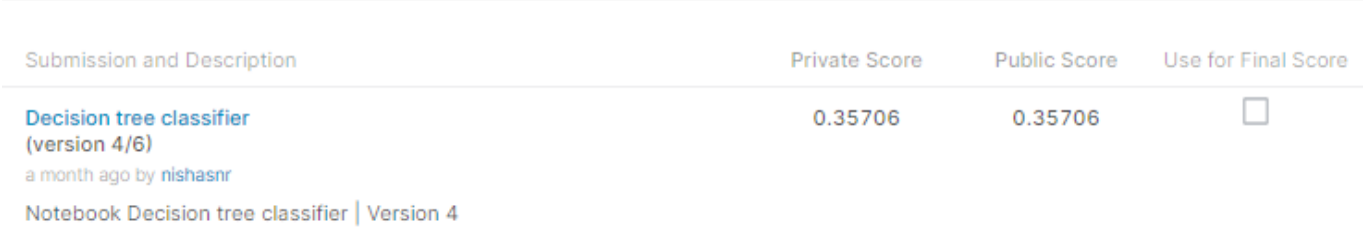


Fig: Confusion Matrix for DTC validation split

The confusion matrix shows the highest value for class 4, this may be because class 4 accounts for most of the data points.
The values of the classification report for the training set look promising. However, the perfect score for different evaluations on the train split may suggest overfitting of the data.
Therefore, we implemented the model for the test dataset and submitted the predictions to the competition. The result was as follows:



This score would put us at position 454 (out of 617) on the leaderboard, i.e. 73.58%.

Conclusion
The near perfect score for the train dataset and the low score for the evaluation and test proves that the decision tree classifier was overfitting on the dataset. This is one of the main disadvantages of this model as they tend to create over-complex trees that do not generalise the data well. Furthermore, they can be very unstable, as slight change in data can lead to formation of a completely different tree.

Model 2.a: Random Forest Classifier

Overview
Random Forest is built by combining a large number of individual decision tree classifiers operating as an ensemble. This means that each of the constituent decision trees give a prediction for the class and then voting is performed and the class with the most votes from the decision trees becomes the prediction for the random forest classifier. Therefore, there needs to be a low correlation between the different decision trees to ensure that the ensemble prediction is better than individual elements. This way, even when some trees are wrong, the majority will be right so the collective result will be right.

Motivation
Random Forest Classifier fits a number of decision tree classifiers on various sub-samples of the dataset which helps control over-fitting. Since the decision tree classifier was overfitting, this model can help in controlling it. Additionally, a decision tree considers every possible feature and picks the one that produces the most separation between the observations in the left node vs. those in the right node. On the other hand, in a random forest classifier, each tree can pick only from a random subset of features. This ensures variation amongst the trees in the model, consequently resulting in lower correlation across trees and more diversification.

Experiments and Results

The train dataset was created by splitting in a 70:30 ratio, the data point corresponding to the household heads.

Firstly, the default values were used for the model with n_estimators = 100

The results are as follows:

	precision	recall	f1-score	support
1	1.00	1.00	1.00	510
2	1.00	1.00	1.00	1138
3	1.00	1.00	1.00	851
4	1.00	1.00	1.00	4190
accuracy			1.00	6689
macro avg	1.00	1.00	1.00	6689
weighted avg	1.00	1.00	1.00	6689

	precision	recall	f1-score	support
1	0.11	0.01	0.02	75
2	0.20	0.01	0.01	141
3	0.07	0.01	0.02	86
4	0.67	0.98	0.80	590
accuracy			0.65	892
macro avg	0.26	0.25	0.21	892
weighted avg	0.49	0.65	0.53	892

Fig: Classification report for RFC training split and validation split

The confusion matrix for the validation set is as follows:

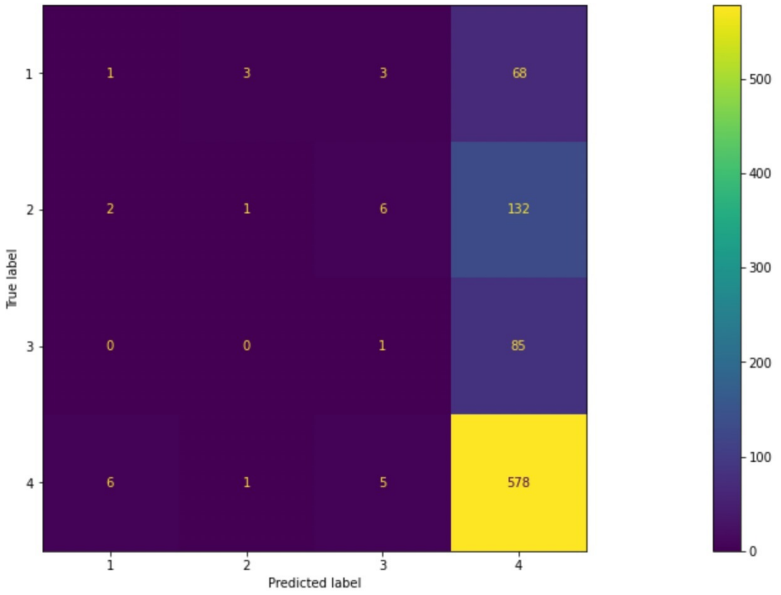


Fig: Confusion matrix for RFC evaluation split

Conclusion

From the results, it is clear that the random forest classifier is still overfitting the data. It may also be because of high correlation between the decision trees since the confusion matrix is also very similar. Other boosting methods might help combat overfitting by using different weak classifiers.

Model 3: Boosting

Model 3.a: Adaptive boosting

Overview

Adaptive boosting is a popular ensemble modeling technique for classification. It combines multiple decision stumps to give the final output. First, a model is built from the training data. The subsequent model then tries to then correct the errors of the previous model by increasing the weight of misclassified points. This process is continued till the complete training set is correctly predicted or maximum number of trees are added. The final prediction is a weighted majority vote

Motivation

Adaptive boosting tends to be less prone to overfitting- probably because the parameters are not optimised jointly. As it is less susceptible to overfitting, it is often referred to as one of the best 'out-of-the-box' classifiers. Moreover, it can build a strong classifier from multiple weak ones. Due to these reasons, adaptive boosting was attempted for this problem

Experiments and Results

A 70-30 split on the data records corresponding to household heads was performed. Scikit-learn was used to build the model

The following parameters were initially used

Parameter	Value
n_estimators	50
learning_rate	1.0

The metrics obtained for training and evaluation are listed below:

Macro F1 score on train set: 0.421

Macro F1 score on evaluation set: 0.34

	precision	recall	f1-score	support
1	0.13	0.06	0.09	63
2	0.37	0.32	0.34	136
3	0.24	0.07	0.11	107
4	0.74	0.90	0.81	586
accuracy			0.65	892
macro avg	0.37	0.34	0.34	892
weighted avg	0.58	0.65	0.61	892

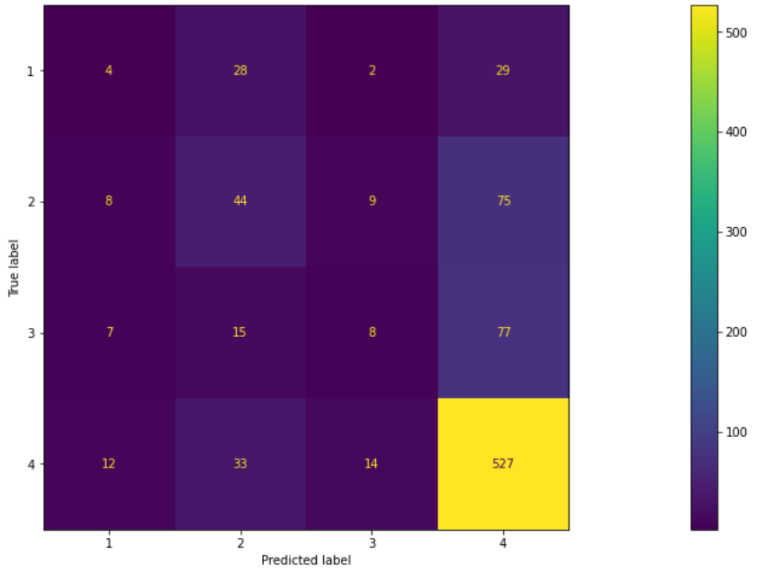


Fig: Classification report and Confusion matrix obtained after evaluation

Hyperparameter tuning was performed using GridSearchCV and StratifiedKFold. It was done on the following combinations of parameters:
n_estimators: 10,50,100,500
learning_rate: 0.0001, 0.001, 0.01, 0.1, 1.0
The following parameters were found to perform the best:

Parameter	Value
n_estimators	500
learning_rate	1.0

After repeating the training and evaluation process using these parameters, the following metrics were obtained:
Macro F1 score on train set: 0.48
Macro F1 score on evaluation set: 0.37

	precision	recall	f1-score	support
1	0.19	0.21	0.20	63
2	0.33	0.26	0.29	136
3	0.19	0.21	0.20	107
4	0.77	0.79	0.78	586
accuracy			0.60	892
macro avg	0.37	0.36	0.37	892
weighted avg	0.59	0.60	0.59	892

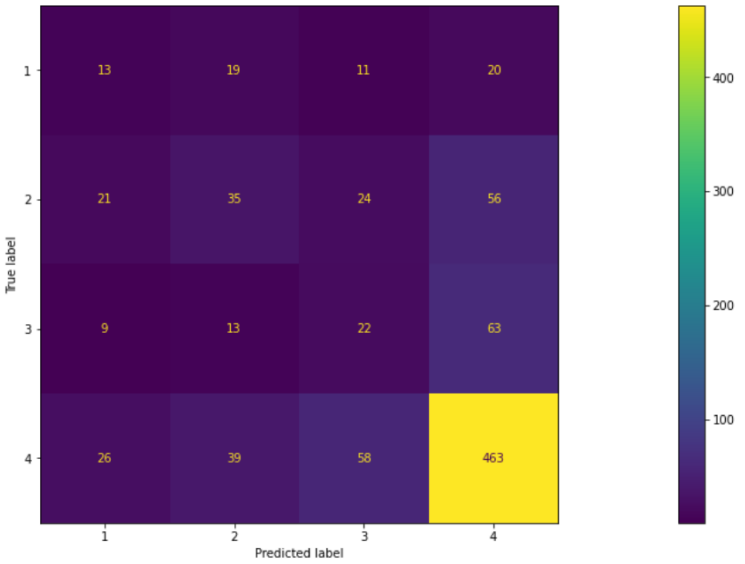


Fig : Evaluation metrics and confusion matrix after hyperparameter tuning

Conclusion
It can be observed that adaptive boosting has a low score for training (as compared to decision tree and random forest classifiers) as well as evaluation. This could be due to the existence of outliers in the dataset as they can affect the performance of adaptive boosting. A boosting algorithm that decreases bias error might be able to solve the problem

Model 3.b: Gradient boosting

Overview

Gradient boosting is another popular ensemble modelling technique for classification that aims to build a strong learner from several weak ones. The weak learners (decision trees) are constructed in a greedy manner and are built based on the previous classifier's residuals thus capturing the variance in the data. This process is continued till the complete training set is correctly predicted or maximum number of trees are added. The final prediction is a weighted majority vote where each tree has a particular weight depending on its accuracy.

Motivation

Gradient boosting is generally used when one aims to reduce bias error. As the macro F1-score achieved by adaptive boosting after training is low, we attempt to first increase that score and then address the overfitting issue

Experiments

A 70-30 split on the data records corresponding to household heads was performed.
Scikit-learn was used to build the model
The following parameters were initially used

Parameter	Value
n_estimators	100
learning_rate	0.1

The metrics obtained for training and evaluation are listed below:
Macro F1 score on train set: 0.7234
Macro F1 score on evaluation set: 0.34

	precision	recall	f1-score	support
1	0.24	0.10	0.14	63
2	0.38	0.27	0.32	136
3	0.20	0.07	0.10	107
4	0.74	0.92	0.82	586
accuracy			0.66	892
macro avg	0.39	0.34	0.34	892
weighted avg	0.58	0.66	0.61	892

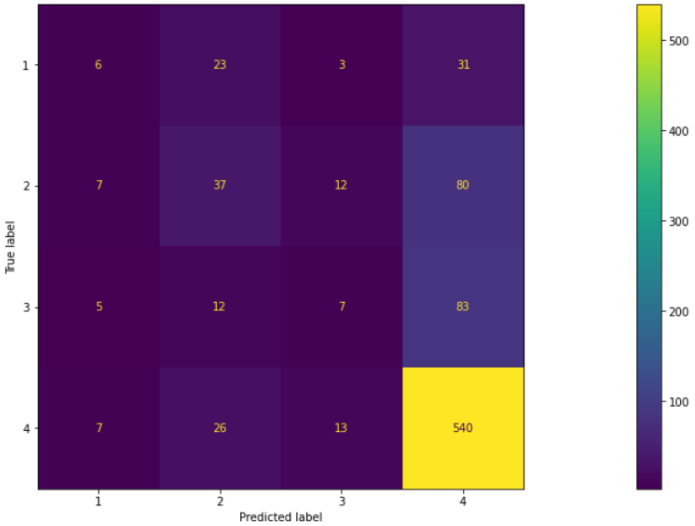


Fig : Evaluation metrics and confusion matrix

Hyperparameter tuning was performed using GridSearchCV and StratifiedKFold. It was done on the following combinations of parameters:
n_estimators: 10,50,100,500
learning_rate: 0.0001, 0.001, 0.01, 0.1, 1.0
The following parameters were found to perform the best:

Parameter	Value
n_estimators	500
learning_rate	0.1

After repeating the training and evaluation process using these parameters, the following metrics were obtained:
Macro F1 score on train set: 0.971
Macro F1 score on evaluation set: 0.37

	precision	recall	f1-score	support
1	0.24	0.14	0.18	63
2	0.33	0.29	0.31	136
3	0.26	0.13	0.17	107
4	0.76	0.88	0.82	586
accuracy			0.65	892
macro avg	0.40	0.36	0.37	892
weighted avg	0.60	0.65	0.62	892

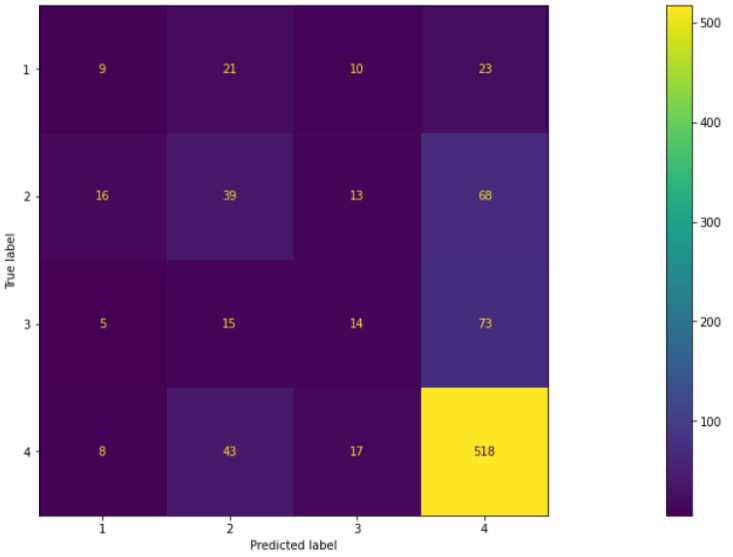


Fig : Evaluation metrics and confusion matrix after hyperparameter tuning

Conclusion
It can be observed that the gradient boosting classifier is overfitting this dataset. As each tree tries to predict the error left over by the previous model, it is possible for gradient boosting to overfit. Other models need to be explored to combat this issue

Model 4: Stacking

Overview

Stacking is an ensemble technique that involves combining the predictions of multiple different models on the same dataset

The architecture of a stacking model involves two or more base models, often referred to as level-0 models, and a meta-model that combines the predictions of the base models, referred to as a level-1 model. Logistic regression is commonly used as the meta-model for classification problems

Motivation

It is often a good idea to use a range of models that make different assumptions about how to solve the task such as decision trees, linear models, support vector machines, neural networks and others. By combining predictions of the models that learn different parts of the problem, it is possible to improve overall performance.

A stacking model combining knn classifier and decision tree classifier as the base model and logistic regression as the meta learner was used to model this problem. This was done to check if the performance of the decision tree classifier could be boosted using stacking.

Experiment and Results

K-fold cross validation was performed to obtain the macro-f1 score on the individual base models as well as the ensemble itself. The results are as follows:

```
>knn 0.338 (0.029)
>cart 0.334 (0.029)
>stacking 0.234 (0.017)
```

Fig: Macro F1 score after cross validation

The household records were then split into train and validation sets in a 70-30 ratio. The metrics obtained are as follows:

0.3356166748669578

	precision	recall	f1-score	support
1	0.67	0.03	0.06	63
2	0.27	0.10	0.15	136
3	0.00	0.00	0.00	107
4	0.68	0.98	0.80	586
accuracy			0.66	892
macro avg	0.40	0.28	0.25	892
weighted avg	0.54	0.66	0.56	892

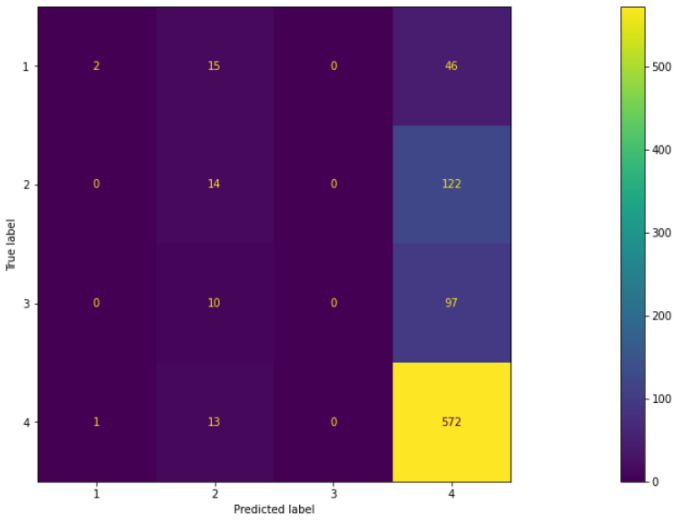


Fig: Evaluation metrics and Confusion matrix obtained after evaluation

Conclusion

The low macro f1 score obtained from stacking is probably because the base models do not perform well independently. This follows from the observation that the decision tree classifier was overfitting the dataset and the fact that the cross-validation score of the decision tree classifier was low. Stacking is effective when the base models perform well and are sufficiently uncorrelated in their predictions.

Model 5: LightGBM

Overview
LightGBM, short for Light Gradient Boosting Machine, is a gradient boosting framework for machine learning originally developed by Microsoft. It is based on decision tree algorithms and used for ranking, classification and other machine learning tasks. The development focus is on performance and scalability.

Motivation
The LightGBM framework supports different algorithms including GBT, GBDT, GBM, and RF. LightGBM has many of XGBoost's advantages, including parallel training, multiple loss functions, regularisation, bagging, and early stopping. A major difference between the two lies in the construction of trees. LightGBM does not grow tree level-wise — row by row — as most other implementations do. Instead it grows trees leaf-wise. It chooses the leaf it believes will yield the largest decrease in loss. Besides, LightGBM does not use the widely-used sorted-based decision tree learning algorithm, which searches the best split point on sorted feature values, as XGBoost or other implementations do. Instead, LightGBM implements a highly optimised histogram-based decision tree learning algorithm, which yields great advantages on both efficiency and memory consumption. The LightGBM algorithm utilises two novel techniques called Gradient-Based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) which allow the algorithm to run faster while maintaining a high level of accuracy.

Experiment and Results
The LightGBM model was used on the training set with the following parameters:
class_weight='balanced', colsample_bytree=0.93, learning_rate=0.01, max_depth=9, metric='None', n_estimators=2000, n_jobs=4, num_leaves=21, objective='multiclass', subsample=0.96

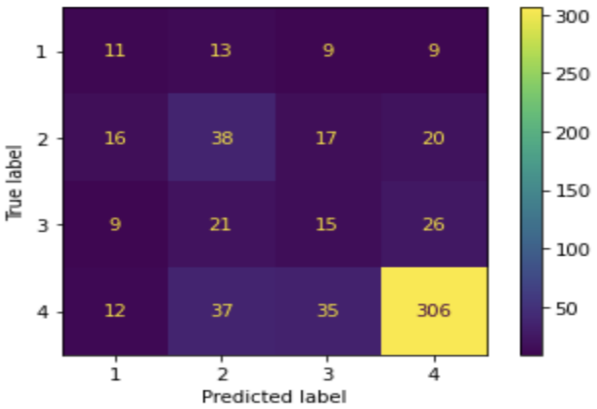
The reason for choosing few of the major parameters values as above are listed below:

1. **Class_weight = ‘Balanced’** : The Class weight parameters are weights associated with classes. Since our task was a multi-class classification, The value was set to ‘balanced’ as this mode uses the values of y to automatically

adjust weights inversely proportional to class frequencies in the input data as *n_samples / (n_classes * np.bincount(y))*

2. **Objective = 'multiclass'** : To specify the learning task and the corresponding learning objective or a custom objective function to be used. Default: 'regression' for LGBMRegressor, 'binary' or 'multiclass' for LGBMClassifier, 'lambdarank' for LGBMRanker.
3. **Max_depth = 9** and **num_leaves = 21**: These values for these 2 parameters were selected after multiple trial runs as the macro-average f1 score was the highest for this pair of all the other tested values
4. **N_jobs = 4** : Number of parallel threads to use for training was set to 4 to fasten the learning process by parallelizing and using GPU.

	precision	recall	f1-score	support
1	0.23	0.26	0.24	42
2	0.35	0.42	0.38	91
3	0.20	0.21	0.20	71
4	0.85	0.78	0.81	390
accuracy			0.62	594
macro avg	0.41	0.42	0.41	594
weighted avg	0.65	0.62	0.63	594



However, even after carefully selecting the training hyperparameters the lightGBM model was overfitting which was observed from the following train and validation accuracies:

```
LightGBM Model accuracy score: 0.6229
Training set score: 0.9029
Validation set score: 0.6229
```

To combat overfitting, we decided to use regularisation and dropout techniques along with early stopping but the validation accuracy score did not improve much which led to us believing that the values for our hyperparameters might not be correct and requires fine tuning. Thus, we decided to implement hyperparameter tuning using GridSearchCV and randomSearchCV

Hyperparameter Tuning

After carrying out hyperparameter tuning the following values were observed to be giving the best training accuracy:

max_depth=-1, learning_rate=learning_rate_power_0997, objective='multiclass', random_state=314+i, silent=True, metric='None', n_jobs=4, n_estimators=5000, class_weight='balanced','colsample_bytree': 0.88, 'min_child_samples': 90, 'num_leaves': 25, 'subsample': 0.94, 'reg_lambda': 0.5

Where new hyperparams like reg_lambda, min_child_samples, subsample were introduced and values for hyperparams like n_estimators, num_leaves,max_depth and random_state were changed.
The learning rate was also changed from a static value to dynamic by introducing a *learning_rate_power_0997 function* that calculates the learning rate after each iteration.

After Hyperparameter tuning the results were better with the macro-average f1 score for the validation set increasing by 1.5%

```
Training until validation scores don't improve for 700 rounds.
[200]   train's macroF1: 0.909718       valid's macroF1: 0.395256
[400]   train's macroF1: 0.949975       valid's macroF1: 0.404862
[600]   train's macroF1: 0.964671       valid's macroF1: 0.414936
[800]   train's macroF1: 0.978034       valid's macroF1: 0.420462
[1000]  train's macroF1: 0.987887       valid's macroF1: 0.428687
[1200]  train's macroF1: 0.995945       valid's macroF1: 0.434488
[1400]  train's macroF1: 0.99801        valid's macroF1: 0.425488
[1600]  train's macroF1: 0.999477       valid's macroF1: 0.421779
[1800]  train's macroF1: 0.999477       valid's macroF1: 0.419201
Early stopping, best iteration is:
[1205]  train's macroF1: 0.995945       valid's macroF1: 0.435804
```

Despite the encouraging results, this f1-score could only get us to the top 30% in the leaderboard.

Voting Classifiers

To try and break into the top 10% we decided to use 3 different LightGBM models with 3 different Voting classifiers knowing that even though there would only be a small increment in the f1-score, it is important to try and check if there is any improvement.
First, only a single LightGBM model was used to train and classify, followed by the model using voting hyperparameter as ‘soft’ which means that that it will average the class with highest probabilities on all individual classifiers and finally a model with voting classifier as ‘hard’ was used which uses the predicted class labels for majority rule voting. The validation score for models with different voting classifiers were observed to be as follows:

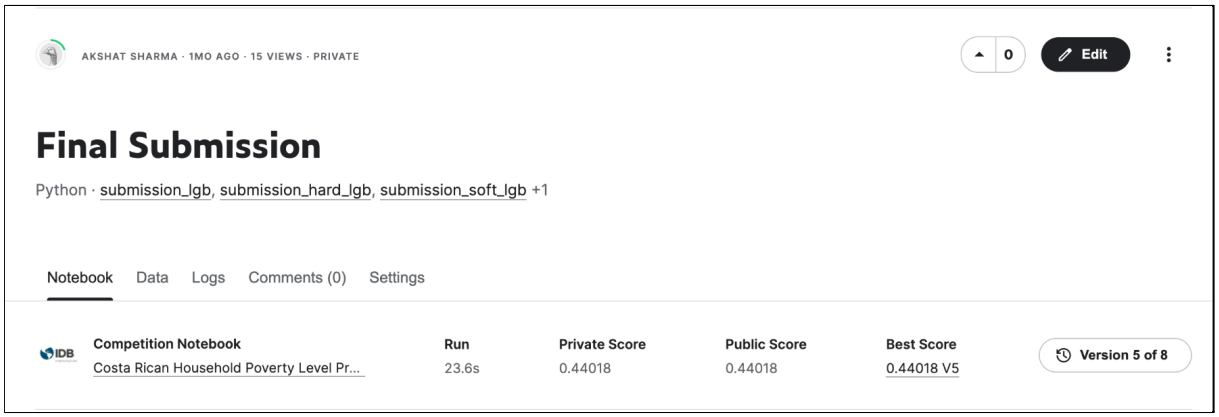
```
Validation score of a single LGBM Classifier: 0.5650
Validation score of a VotingClassifier on 3 LGBMs with soft voting strategy: 0.8889
Validation score of a VotingClassifier on 3 LGBMs with hard voting strategy: 0.8022
```

We decided to test all 3 models to see how they generalise over unseen dataset and submitted the poverty class/target predictions which gave us the following results:

Model	F1 Score (test set)	Public Leaderboard Ranking
Single LightGBM + Hyperparameter Tuning	0.39939	268
LightGBM + Hyperparameter Tuning + Soft Voting Classifier	0.43536	98
LightGBM + Hyperparameter Tuning + Hard Voting Classifier	0.44018	44

7. Score and position on Leaderboard

The model that gave us the best macro F1 score on the test dataset was the **LightGBM model with hyperparameter tuning and Hard Voting Classifier**. We were able to achieve the score of **0.44018**, which placed us at the **44th** position on the Public Leaderboard. There were 617 positions on the leaderboard putting us in the **top 08%**.



Note: After finishing the report and video, we ran the entire code again to make sure the source code is error free and due to lgbm model being trained on random train data instances (after train test split), we found that the accuracy scored increased and upon testing the model again on test set the model '**LightGBM + hyperparameter tuning and Soft voting classifier'** gave a slightly better result than our final submission file. Due to time constraints and video being recorded we decided not to change the report and video so that any discrepancies don't creep up. To facilitate the evaluation we will be submitting csv files for both the initial submission with lgbm, hyperparam tuning and hard voting classifier as well as the csv file generated after video recording and report completion from lgbm, hyperparam tuning and Soft voting classifier.

8. Novelty

1. Chi-Squared Test

Finding the correlation between features and the target is a good way to find important features in the dataset and are necessary in prediction of the target. As mentioned earlier, the data fed into the model vastly affects its performance. Thus, finding the most impactful features can improve the quality of data that is input into the model and consequently improve its performance.

Chi-Squared test of independence was performed on the categorical features of the dataset to find if they had correlation with the target. This gave us very useful results. We found there were 122 features with the p value between the range of 0.0 and 0.05, meaning that there exists some relationship between those features and the target. This helped us in reducing the dimensionality of the feature space as we considered only the top 50 and 100 features that had the most impact on the target and trained the model accordingly. This not only reduces the time and space that are required by the model but also improves the interpretation of the parameters of the model.

2. Hyperparameter tuning

While solving this classification problem, we came across many different techniques,algorithms,models etc. but, most of these techniques were quite common and used by many people participating in this competition. However, one thing which most people didn't consider bothering about was the hyperparameter values. Most of the notebooks submitted on kaggle simply copied the hyperparameter values from a top team and added a reference but we wanted to see for ourselves how the change in hyperparameter values affects the model performance and which set of values would give us an optimal score which is why we decided to do hyperparameter tuning using RandomSearchCV and GridSearchCV.

9. Challenges Faced

The main challenge we faced was with the dataset. The dataset contained unbalanced data values. For example, the four levels of the poverty levels varied from 63 per cent to 8 per cent. In addition, many columns had missing data values. For example, there were 6860 rows with "rent" with NaN values. Moreover, the data in similar columns were inconsistent. For instance, the data values under the columns "age" and "years_behind_school" are inconsistent.

There were also problems with the data column:

1. The names of the columns were illogical. It created ambiguity and needed to be changed entirely.
2. Some data columns were duplicated with slightly different names, which required us to remove them.
3. The dataset contained columns with squared values of other columns, which did not provide additional information.It also had columns that did not give any information for solving the problem.

The other problem with the dataset is that it is not large enough data to apply some deep neural network techniques. We also came across the problem of overfitting. The model learns the weights and biases specific to the training data but does not generalise it. As a result, we obtain high accuracy for the training data, but the accuracy drops a lot when used on the testing data.

10. Lessons learnt

1. Data cleaning and preprocessing

After analysing the data, we recognized the need to clean the data and pre-process it. Without this vital step in the pipeline, issues such as missing and inconsistent values would have been overlooked. This would have subsequently reduced the quality of the model predictions

Through critical thinking, discussion and analysis, we decided which features needed to be engineered to make the household head records more informative. We understood the relevance of correlation, learnt about the chi-square test and then applied these 2 concepts for feature selection.

However, we also recognize that while data cleaning and processing can help improve model training and predictions, it does not guarantee it. There are several other factors to consider such as the data itself (quantity and quality), the problem to be solved and the models used

2. Overfitting issue

As several models performed higher on the training data than the test data, we recognized the problem of overfitting. In order to combat it, we tried feature selection, hyperparameter tuning across different models as well as different types of ensemble. While these techniques did not show an improvement every time they were applied, we were able to achieve our best score by applying a voting classifier on 3 LightGBM models

In short, we practically encountered the problem of overfitting and learnt how to combat it.

3. Different ML models

Through the process of understanding this problem and brainstorming a solution for it, we learnt about different ML models and experimented with them. These include models and ensemble methods such as SVM, decision tree, random forest classifier, adaptive boosting, gradient boosting, stacking and LightGBM

We learnt about the underlying concepts for each model as well as the advantages and disadvantages for each

11. Conclusion

We achieved a macro F1 score of 0.44018 which placed us in the top 10% (45th out of 617) of the public leaderboard. The highest score in the competition is 0.44878 which is not that good.
It can be considered as an indication for the quality of the data provided. Any machine learning model is highly dependent on the provided data. Even after performing exploratory data analysis, preprocessing and feature engineering, the attributes were still not sufficient to predict the poverty classes. Furthermore, the high number of missing values and discrepancies in the data given, raises questions about the quality of the data collected.
It is important to point out that across all the different models we tried, there were some features which were identified by most as important.
Nevertheless, this was a good opportunity in learning how to preprocess and prepare the data for training. It also gives an insight about what to expect from real world datasets which can have erroneous and missing data.
This competition and the results would help the Inter-American Development Bank(IBD) in figuring out what data they should collect and how they can improve the quality of records.

References

[1] Vijay Kotu, Bala Deshpande, in Data Science (Second Edition), 2019
[2] https://www.jmp.com/en_be/statistics-knowledge-portal/chi-square-test/chi-square-test-of-independence.html
[3] https://libguides.library.kent.edu/spss/chisquare