# Assignment-2 Report

## Pre-processing:

- Scaling numerical features (obviously) did not affect performance, hence it was not done. There were no missing values to handle.
- Since we built binary tree, large number of categories of a feature could affect the performance. In particular both the features 'job' and 'month' had 12 categories. These were reduced by clubbing in the following way:
    1) 'job': based on closeness of ratio of 'yes'/'no' as well as nature of job, a new category of white-collar was created comprising 'admin', 'management' & 'technician'; 'services' & 'housemaid' were clubbed under 'blue-collar'; 'entrepreneur' & 'self-employed' clubbed together.
    2) 'month': if consecutive months had very close 'yes'/'no' ratio, they were clubbed together: (jan,feb), (may,june,july, august), (september, october)

  This step indeed increased F1-score & accuracy.

- 'day' had a minute correlation with y and it did not make sense intuitively to be related to the 'y'. Including it increased running time as well as worsened performance. Nevertheless, it was later specified on Piazza that we are not allowed to exclude features.

## Tree building:

- Node class has attributes feature, threshold, left child, right child, value, number of samples in it, impurity( using gini impurity or entropy). It has a method to check if the node is a leaf node. A node is leaf if its value is not None.
- Tree class has attributes min_samples_split, max_depth, min_gain for pre-pruning. Besides that, there is a criterion attribute to select gini or entropy to build the tree. Imbalance is handled manually by assigning class weight to class 'y'='yes' relative to 'y'='no'.
- While calculating entropy/ gini index, the fraction of class 'y'= 'yes' is multiplied by class weight which can be specified before constructing the tree. This helped improve the accuracy as well as f1_score.

- While finding the best split at each node in case of numerical features with significant range, only potential thresholds were considered where y label changes.
- In case of splitting using a numerical feature with threshold y, left child was assigned the samples <=y and right child assigned the other samples.
- For categorical features, the left child was assigned samples with feature value = threshold class and right child assigned the other samples (hence we only check {1, n-1} type of splits). Since we ensured the number of categories in each feature is sufficiently small, the binary tree won't go to very large depth and there is no need to check every possible splitting like done in CART algorithm.
- Cost Complexity calculation: Since there was no clear instruction or definition of how to calculate this, we assumed :

  Cost complexity of a node= Risk of the node + alpha* Number of leaves in subtree

  Risk of the node = Total risk of all leaves in the subtree

  Risk of a leaf node is approximated by (impurity)*(number of samples in leaf)

alpha is an adjustable hyperparameter.

## Pruning & Final Results:

- The dataset was split into 0.7:0.2:0.1 for training, cross validation and testing.
- Pre-pruning was done using the three parameters: depth, samples in leaf & gain. Class weight between 1.5-2.5 gave the best results. After hours of hyperparameter tuning, best results were obtained using min_samples=19, max_depth=20, min_gain = 0.02
- ~89.3% Accuracy, ~0.295 F1-score & ~0.63 Precision was achieved on the validation set.
- Gini criterion outperformed entropy without exception
- A general pattern was that tightening pre-pruning parameters increased Precision & Accuracy and relaxing them increased Recall.
- Post pruning strategy: Bottom-up pruning. If both children of a node are leaves, it will be pruned and then the new tree is tested on validation set. If it does not significantly worsen the performance, pruning is kept otherwise it is undone.
- Performance is quantified by the recall on the CV set, since pre-pruning results in small recall and high precision. We did not use cost-complexity for measuring performance since it gave way worse results.

- Allowing the tree to overfit too much resulted in worsened results in training, cross validation as well as testing after post pruning.
- Post pruning significantly changed the structure (depth and number of leaves) of the tree as well as significantly improved the performance relative to the cross validation set.
- Instead of building a separate test.py, we predicted on the final test set in the notebook itself. First, the necessary pre-processing of the test set was done and then prediction done using the final pre+post pruned tree obtained earlier.

The general effect of post pruning can be seen from this example: taking the optimal pre-pruning parameters as above,

Training F1 score: 0.2823

CV F1_score: 0.2478

Initial depth= 20

Initial number of leaf nodes= 54

Then Post pruning is done guided by cross validation set

Final F1_score: 0.295

Final depth: 7

Final number of leaf nodes: 8