CS771 Assignment 2 : ML Noobs

(200186): Arnav Gupta; (200204): Aryan Vora
(200539): Kushagra Sharma; (200556): Mandar Wayal
(201050): Tejas R

October 2022

1 Part I

We have used **Decision Trees** to classify the error classes (first kind of approach in DTs, as mentioned in the Assignment document). **sklearn** provides in-built effecient implementation of the model, which we have used (Reference: sklearn.tree.DecisionTreeClassifier). We have trained our model on 80% of the data provided and evaluated it on the remaining data. Fine-tuning was done by experimenting with the parameters that the **sklearn** library provides.

We worked with the following **hyperparameters** in the decision tree model:

- **max_depth** : This refers to the maximum depth of the Decision Tree. The depth of the tree directly represents its complexity, which makes one of the most important hyperparamaters to tune.
- **criterion** : This hyperparameter decides the splitting criteria at a node in the decision tree.
- **min_samples_split** : This decides the degree upto which nodes are split in the decision tree. The default value is set to 2, which means that a node will be split only if there are atleast 2 samples at the node. Hence, in the default case, all leaves contain only 1 sample.
- **min_samples_leaf** : Using this hyperparameter, we can specify a lower bound on the number of samples present at a leaf node.
- **splitter** : It decides between the 2 strategies to split a node : "best" and "random". If splitter is set to "best", the decision tree algorithm chooses the subset of features which gives the best split. If it is set to "random", a random set of features is taken.
- **max_leaf_nodes** : With these, specify an upper bound on the total number of leaf nodes in the decision tree.

The process of **tuning** these hyperparameters is explained below :

- Firstly, the **max_depth** parameter was varied in the model. We tried the values (5, 10, 15, 20, 25, 30, 40, 50) while keeping the other parameters constant or default. It was observed that the metrics showed improved performance till max_depth=20. From max_depth=30, the model did progressively worse on the test set (albeit showed a constant or improved performance on the training set). This can be attributed to the decision tree overfitting the train data in the case of high max_depths. After some more fine tuning by trying all max_depths from 20 to 30, max_depth=22 gave the best results.
- We tuned **min_samples_split** next. After taking the values (1, 2, 3, 5, 10, 20), it was found that the model kept improving on the metrics. Upon some more fine-tuning, we arrived at the optimal value of 27. A reason for this observation may be that leaf nodes having less samples may lead to overfitting, since the decision tree may become more complex due to the greater splits required to make the leaf nodes sufficiently small.
- Next, we decided on the **criterion** of the model (i.e. "gini" or "entropy"). After training multiple decision trees using default and tuned parameters, it was found that "gini" almost always performed better on the test set. Hence, we used this criterion.

- Now, for the **splitter**, it was found that splitter="random" gave a large variance even when the same decision tree was run multiple times using the same hyperparameters. This was expected due to the random choice of features. We set it to "best", since although "random" sometimes gave better results, the improvement was not enough to justify the added uncertainty of the predictions.
- The other 2 parameters, **min_samples_leaf** and **max_leaf_nodes** were not able to give any improvement upon tuning. Hence, we used the default value of min_samples_leaf(=1) and did not include max_leaf_nodes in the parameters passed to the model, since doing so decreased the performance of the model.

The **final hyperparameters** used were **max_depth**=30, **criterion**="gini", **min_samples_split**=27, **min_samples_leaf**=1 and **splitter**="best".

We also experimented with **OvA** (One versus All) and **LwP** (Learning with Prototypes). There were trade-offs with respect to the precision metrics, model size and training time, as described in Part II.

2 Part sw3e3rr'hII

Let us analyze the difference in the **Decision Tree** model that we employed versus **OvA** and **LwP**

Advantages:

- There are several '**rare**' **classes** which have a very few datapoints. For such classes the data is unbalanced, and the OvA has a lot of datapoints 'against' the class as opposed to 'for' the class. This gives a very poor estimate (i.e. classifier) for the class, which in-turn leads to a lower ***mprec@k***. However, a decision tree chooses the node action suitably well to classify these datapoints well.
- A decision tree (once trained), can classify test data-points much faster compared to the OvA classifier, giving a lower **prediction time**. Since a training point has to traverse an almost-balanced tree (a good DT) only once to get classified, while it has to compute the value in all 'C' classifiers in OvA.
- A decision tree requires less **pre-processing** of data to train compared to its OvA counterpart, thereby making it easier to code.
- The decision tree, with well-tuned hyperparameters, gave a much better overall accuracy,***prec@k*** ,sas opposed to OvA.

Disadvantages:

- LwP gave better ***mprec@k*** as it performed better on the rare instances of error classes.
- The **model size** of the Decision tree model is far larger than the LwP model or the OvA model.
- A decision tree if grown excessively can easily **overfit** the data, which is why the hyperparameters must be tuned carefully to prevent this.