**Hyperparameters Used:**

* **max\_depth**
  + it controls the depth of each decision tree in the ensemble, as deeper trees tend to overfit, this parameter can be used to avoid overfitting.
* **min\_samples\_leaf**
  + A smaller leaf makes the model more prone to capturing noise in train data.
* **min\_samples\_split**
  + This parameter ensures that a split is done only when the node will has at least the number of samples specified in ‘min\_samples\_split’.
  + This parameter helps in overcoming the overfitting problem by restricting splits when samples are less than ‘min\_samples\_split’ in a given node.
* **max\_features**
  + The maximum number of Features that are evaluated for splitting at each node.
  + Increasing max\_features generally improves the performance of the model as at each node now we have a higher number of options to be considered. However, this is not necessarily true as this decreases the diversity of individual tree which is the USP of random forest. But, for sure, you decrease the speed of algorithm by increasing the max\_features. Hence, you need to strike the right balance and choose the optimal max\_features.

* **n\_estimators**
  + Number of Decision Trees to be built in the Forest.
  + Generally more number of decision trees in a random forest, the better generalization will be, but after a certain number of trees, our model becomes computationally costly, hence we need to find optimum n\_estimators.
  + Higher number of trees give you better performance but makes your code slower. You should choose as high value as your processor can handle because this makes your predictions stronger and more stable.

**Use of Cross-Validation**

***cross-validation*—**to do a sequence of fits where each subset of the data is used both as a training set and as a validation set.

As we train any model on the training set, it tends to overfit most of the time, and in order to avoid this situation, we use regularization techniques. Cross-validation provides a check on how it is performing on a test data (new unseen data), and since we have limited training instances, we need to be careful while reducing the amount of training samples and reserving it for testing purpose.

The best way to improve the performance of the system without compromising much would be to use a small part of the training data itself to validate, as it might give us an idea of the model's ability to predict unseen data.

**k-fold is a popular kind of cross-validation technique**, in which, say k=10 for example, 9 folds for training and 1 fold for testing purpose and this repeats unless all folds get a chance to be the test set one by one. This way, it provides a good idea of the generalization ability of the model, especially when we have limited data and can't afford to split into test and training data.

**how you have obtained the best set of hyperparameters?**

I used Sklearn’s GridSearchCV to find the optimal hyperparameters using following parameters

param\_grid = {

'max\_depth': [3,8,1],

'min\_samples\_leaf': range(100, 400, 100),

'min\_samples\_split': range(200, 500, 100),

'n\_estimators': [100,1000, 150],

'max\_features': [5, 10, 1]

}