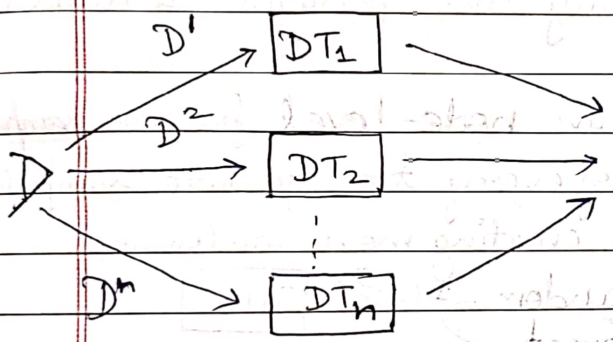
Random Forests

Random Forests are a popular ensemble learning method which uses **decision trees** as the base models under the hood for performing both classification and regression tasks. They operate by creating multiple decision trees on different subsets of data used for training multiple trees together in **parallel**.

→ For classification task, we take the majority vote for predictions made by individual trees.

→ For regression task, we take the mean of values predicted by individual decision trees.



Classification	Regression
Mode of all the predictions	Mean of all the predictions.

Bootstrap Aggregation → To train each decision tree, random forests use **bagging**, where each tree is trained on a different random subset of the data (created by sampling with replacement). This introduces diversity among the trees, reducing the likelihood that they all make the same error.

Random Feature Selection → In addition to random sampling of the data, random forests introduce further randomness by selecting a random subset of features at each split in the tree. This prevents any single feature from dominating the model and helps in handling high dimensional data.

Advantages: (1) Reduces overfitting by averaging results of multiple trees, random forests tend to generalize better.

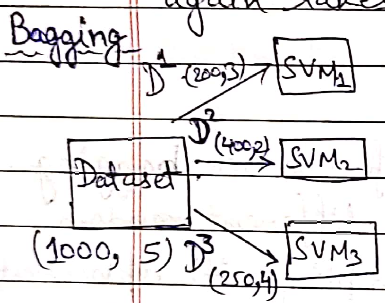
(2) The model becomes less sensitive to noisy data and outliers due to the diversity of the trees.

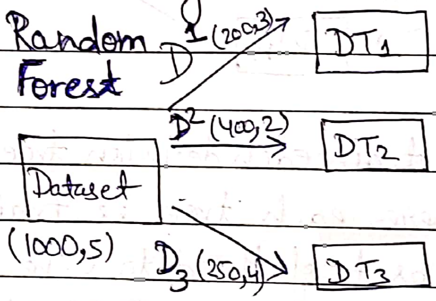
Difference between bagging and random forest

Although Random Forests are a kind of bagging algorithm themselves but they are a bit different from standard bagging algorithms. Two major distinctions are—

(a) Bagging algorithms can use any of the algorithms like (decision trees, KNN, SVM), etc. as the base models whereas Random Forests can only use decision trees as their base model.

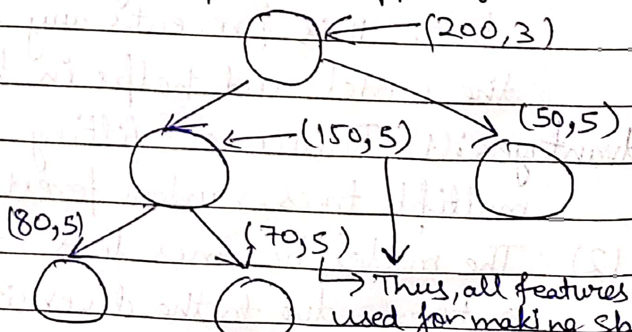
(b) In Bagging, we have **tree-level** feature sampling meaning feature subsets from data is created before making the tree and only these features are available for making a split.
In Random forests, we have **node-level** feature sampling meaning at each node of every tree feature sampling again takes place thus creating more randomness.





→ While making individual ~~decisi~~ SVM models only 3 features are used in SVM_1, 2 in SVM_2 & 4 in SVM_3 at every node.

→ While making individual decision tree models in the first DT_1 model only 3 features will get used from data but in the next node all samples will be considered for making next split. Similar process happens for all DTs.



→ Thus, all features are used for making splits again in Random Forest.

Hyperparameters in Random Forest Algorithm :

(1) n-estimators → The number of trees in the forest. default = 100.

(2) criterion → The function to measure quality of a split. Supported criterions are (gini, entropy and log-loss).

(3) max-depth → The maximum depth for the tree. If none, then nodes are expanded until all leaves are pure or until all leaves contain less samples than "min-samples-split" sample.

(4) min-samples-split → The minimum number of ~~l~~ leaves required to split an internal node, default = 2.

(5) min-samples-leaf → The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min-samples-leaf training samples in each of the left and right branches.

(6) max_features → The number of features to consider when looking for best split. If int, then considers max-features at each split. Other available options are sqrt, log2 and None. If sqrt, then max-features = sqrt(n-features), if log2, then max-features = log2(n_features). If None, then max_features = n-features.

(7) bootstrap → Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build whole tree. By default = True

(8) max_samples → If 'bootstrap' is True, the number of samples to draw from X to train each base estimator.

- If None (default), then draw X.shape[0] samples, i.e., all.
- If int, then draw max-samples samples.
- If float, then draw that much percent samples.

(9) ccp-alpha → Complexity parameter used for minimal cost complexity pruning. By default, no pruning is performed.

(10) oob_score (11) min-weight-fraction-leaf (12) min-impurity decrease
(13) n_jobs (14) random-state (15) class_weight (16) warm_start