Classifier

Classifier Training

It refers to the process of teaching ML model to make predictions or decisions based on input data. Steps involved:

1. Input Data
2. Feature Extraction
3. Splitting Data
4. Choosing Model
5. Training Model
6. Evaluation
7. Hyperparameter Tuning
8. Deployment

Gaussian Mixture Model (GMM)

It is Clustering Algorithm used to organise data by identifying commonalities and distinguishing them from one another. The GMM is trained using the Expectation-Maximisation (EM) algorithm, an iterative approach for determining the most likely estimations of the mixture Gaussian distribution parameters. The EM method first makes rough guesses at the parameters, then repeatedly improves those guesses until convergence is reached.

Algorithm:

- **Initialise phase:** Gaussian distributions' parameters like covariances and mixing coefficients should be initialised
- **Expectation phase:** Determine the likelihood that each data point was created using each of the Gaussian distributions.
- **Maximisation phase:** Apply the probabilities found in the expectation step to re-estimate the Gaussian distribution parameters.
- **Final phase**: To achieve convergence of the parameters, repeat steps 2 and 3.

One-Vs-All (OVA) Multi-Class Classification

Each binary classifier is trained to distinguish instances of a specific alert type from all other instances (all other alert types combined). Hence, we have a binary classifier for each alert type.

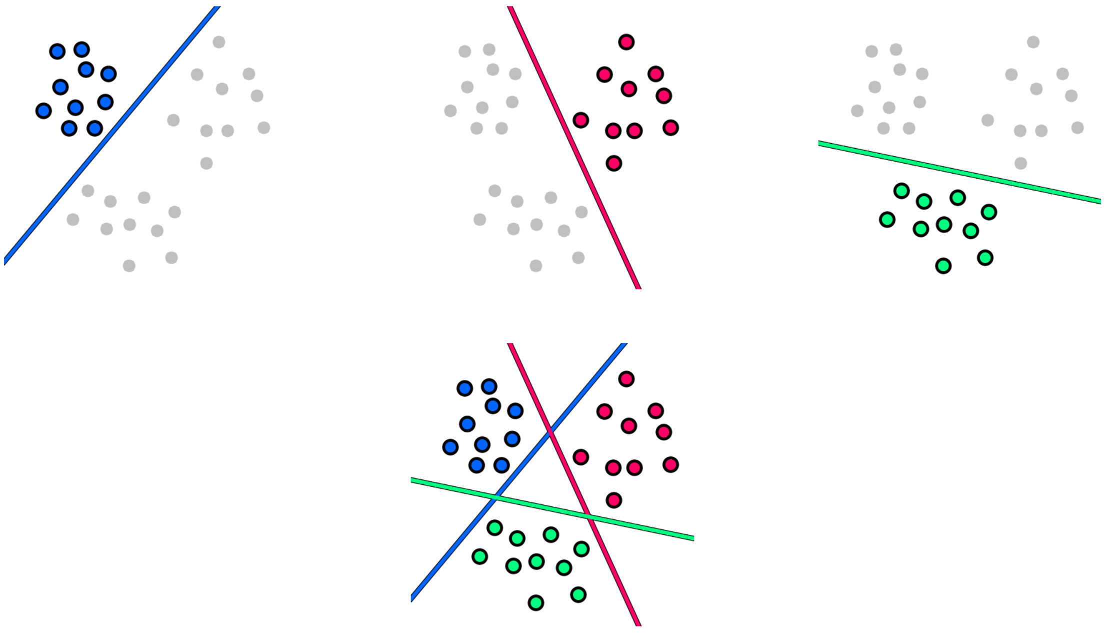
Training OvA Classifier

We learn C two-class classifiers on the entire dataset, with the Cth classifier trained to distinguish the Cth class from the remainder of the data. With the Cth two-class subproblem we simply assign temporary labels yp to the entire dataset, giving +1 labels to the Cth class and −1 labels to the remainder of the dataset

$(y_p)' = +1$ if y_p = C, else -1

Positive Side of Single Classifier

The single class being distinguished is coloured with its original colour with the corresponding learned decision boundary coloured similarly and all other data is coloured grey.



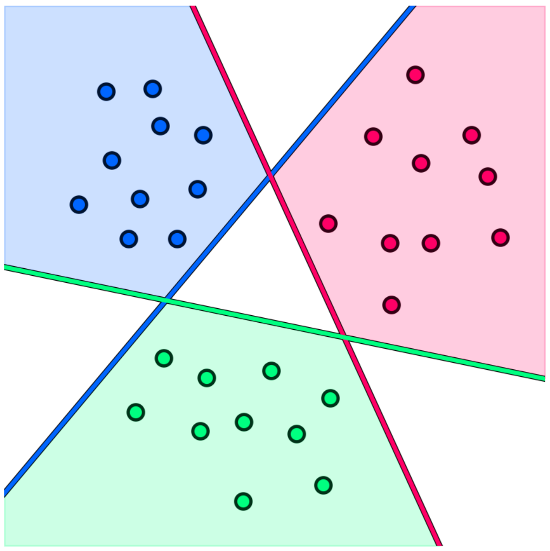
Denoting weights of Cth classifier as w_c and corresponding decision boundary as $x^T w_c = 0$

For jth classifier, we have for pth point xp,x_p,$x^T w_c > 0$ if $y_p = C$, else < 0

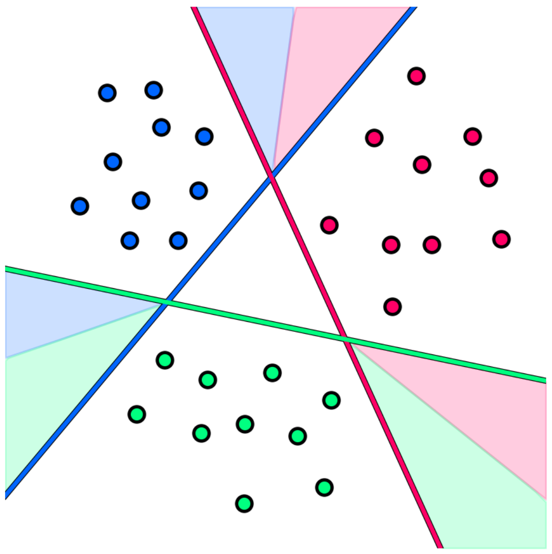
This implies that when evaluated by each two-class classifier individually, the one identifying a point's true label always provides the largest evaluation, i.e., $x^T w_j = max(x^T w_c)$ for all c from 0 to C-1

Therefore to get the associated label y, we therefore want the *maximum argument* of the right hand side, since this gives the index associated with the largest classifier. So,

$y = argmax(x^T w_c)$ for all c from 0 to C-1



Positive Side of Multi-classifier



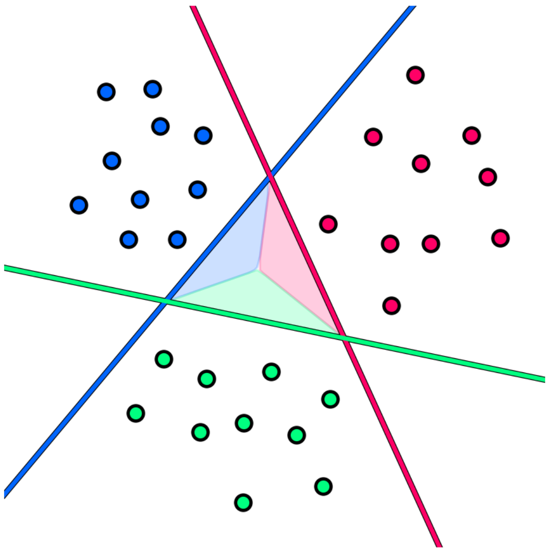
We assign a class label at random for points equidistant to two or more decision boundaries. These points form the *multi-class decision boundary*.

The *signed distance* of an arbitrary point x to the jth linear decision boundary is equal to the evaluation at this boundary *provided we normalize its parameters by the magnitude of its feature-touching weights.* Hence, the signed distance of x to jth boundary = $x^T w_j / ||w_j||$

So, normalized weight vector is $w_j N = w_j / ||w_j||$

Negative Side of All Classifiers

Every point in the region lies on the negative side of our classifiers and all signed distances are negative. Hence the shortest distance *in magnitude* is the largest *signed* distance, being the smallest (in magnitude) negative number. Conversely the largest distance *in magnitude* is the largest (in magnitude) negative number, and hence the smallest *signed* distance



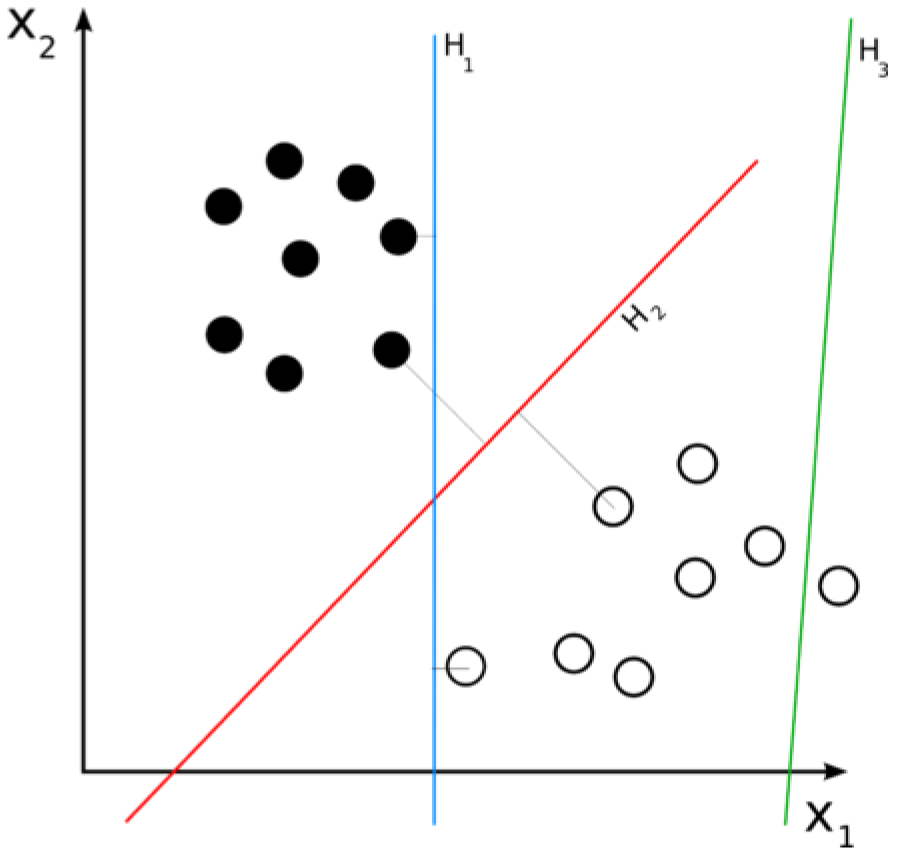
LinearSVC

LinearSVC is used for linear classification problems, where the goal is to find a hyperplane that separates the input data into different classes.

It is the base classifier for the OVA approach is a Linear Support Vector Machine (LinearSVC). LinearSVC is commonly used for binary classification tasks, and in the OVA, it creates multiple binary classifiers, one for each class. So, $y = wx + b$ where w is weight function and b is bias

Regularisation: Controlled by hyper-parameter C, determining the trade-off between achieving a smooth decision boundary and accurately classifying training data.

Loss Function: Uses hinge loss function for optimisation, which penalises misclassifications



Random Sampling

- Oversampling - Duplicating samples from the minority class
- Undersampling - Deleting samples from the majority class

Both oversampling and undersampling involve introducing a bias to select more samples from one class than from another, to compensate for an imbalance that is either already present in the data, or likely to develop if a purely random sample were taken.

SMOTE (Synthetic Minority Over-Sampling Technique)

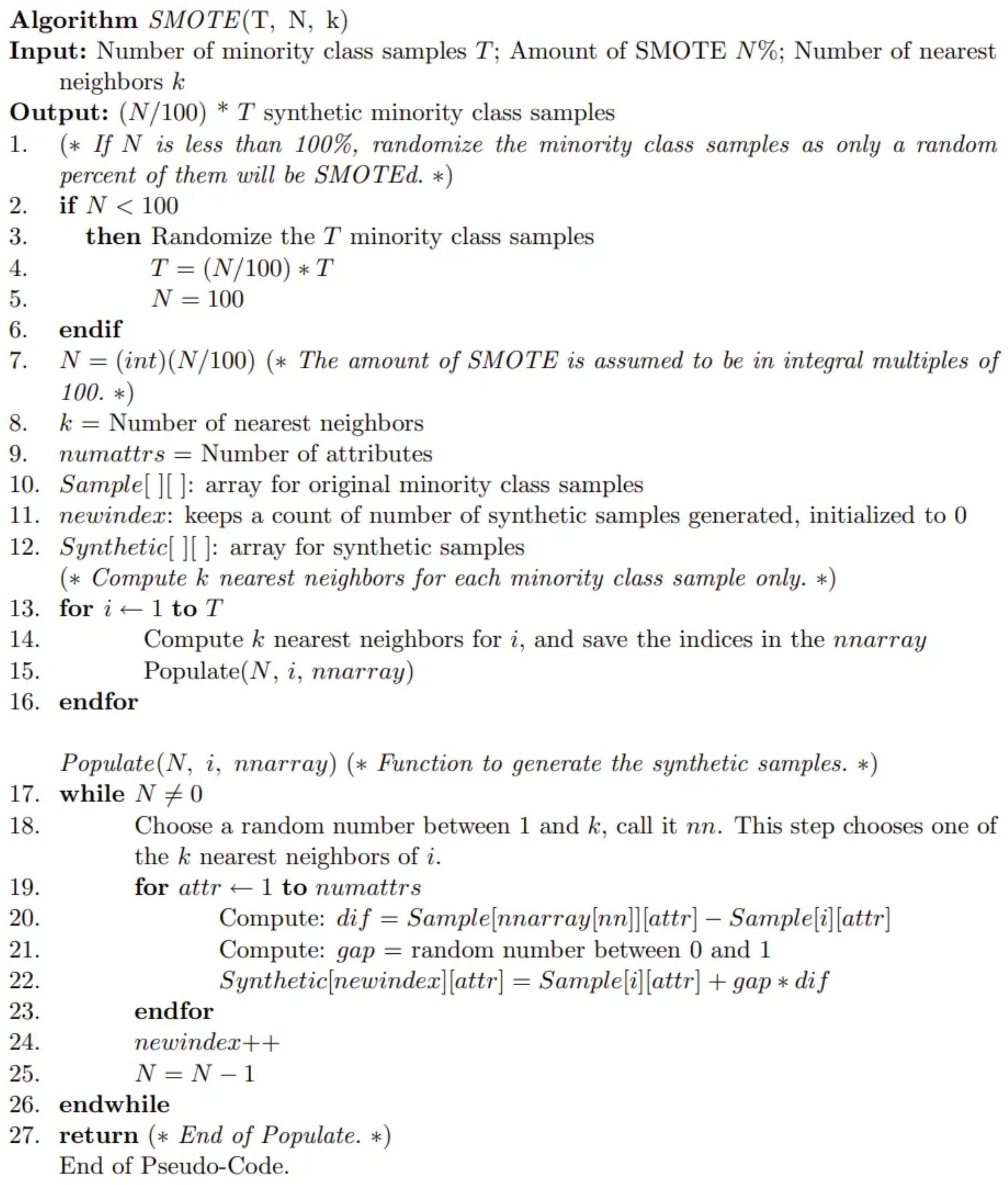
Algorithm:

- Take difference between a sample and its nearest neighbour
- Multiply the difference by a random number between 0 and 1
- Add this difference to the sample to generate a new synthetic example in feature space

- Continue on with next nearest neighbour up to user-defined number

SMOTE first selects a minority class instance a at random and finds its k nearest minority class neighbours. The synthetic instance is then created by choosing one of the k nearest neighbours b at random and connecting a and b to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances a and b.

Mathematical Model of SMOTE:

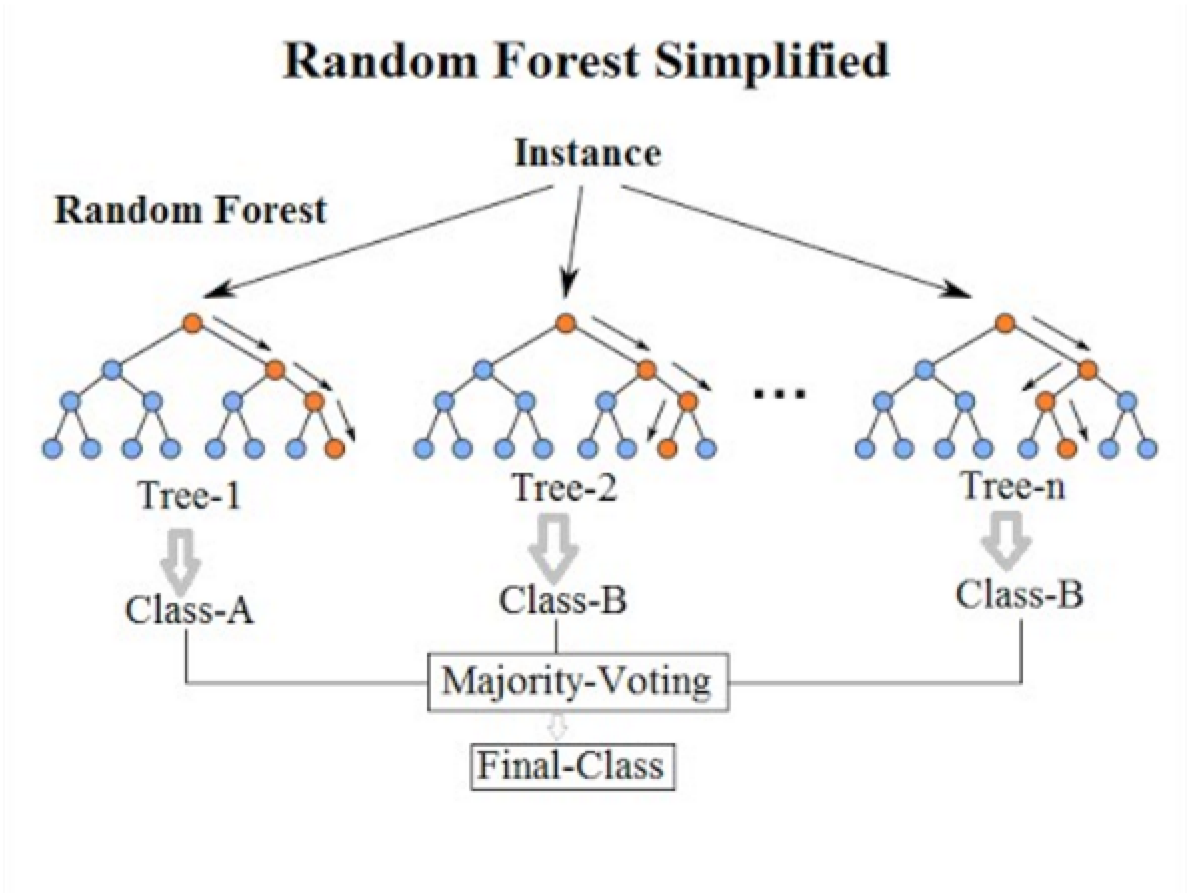


```
Algorithm $SMOTE$(T, N, k)
Input: Number of minority class samples $T$; Amount of SMOTE $N\%$; Number of nearest
    neighbors $k$
Output: $(N/100) * T$ synthetic minority class samples
1.  (* If $N$ is less than 100%, randomize the minority class samples as only a random
    percent of them will be SMOTEd. *)
2.  if $N < 100$
3.    then Randomize the $T$ minority class samples
4.         $T = (N/100) * T$
5.         $N = 100$
6.  endif
7.  $N = (int)(N/100)$ (* The amount of SMOTE is assumed to be in integral multiples of
    100. *)
8.  $k$ = Number of nearest neighbors
9.  $numattrs$ = Number of attributes
10. $Sample$[ ][ ]: array for original minority class samples
11. $newindex$: keeps a count of number of synthetic samples generated, initialized to 0
12. $Synthetic$[ ][ ]: array for synthetic samples
    (* Compute $k$ nearest neighbors for each minority class sample only. *)
13. for $i \leftarrow 1$ to $T$
14.        Compute $k$ nearest neighbors for $i$, and save the indices in the $nnarray$
15.        Populate($N$, $i$, $nnarray$)
16. endfor

    $Populate(N, i, nnarray)$ (* Function to generate the synthetic samples. *)
17. while $N \neq 0$
18.        Choose a random number between 1 and $k$, call it $nn$. This step chooses one of
           the $k$ nearest neighbors of $i$.
19.        for $attr \leftarrow 1$ to $numattrs$
20.               Compute: $dif = Sample[nnarray[nn]][attr] - Sample[i][attr]$
21.               Compute: $gap$ = random number between 0 and 1
22.               $Synthetic[newindex][attr] = Sample[i][attr] + gap * dif$
23.        endfor
24.        $newindex$++
25.        $N = N - 1$
26. endwhile
27. return (* End of Populate. *)
    End of Pseudo-Code.
```

Random Forest Classifier

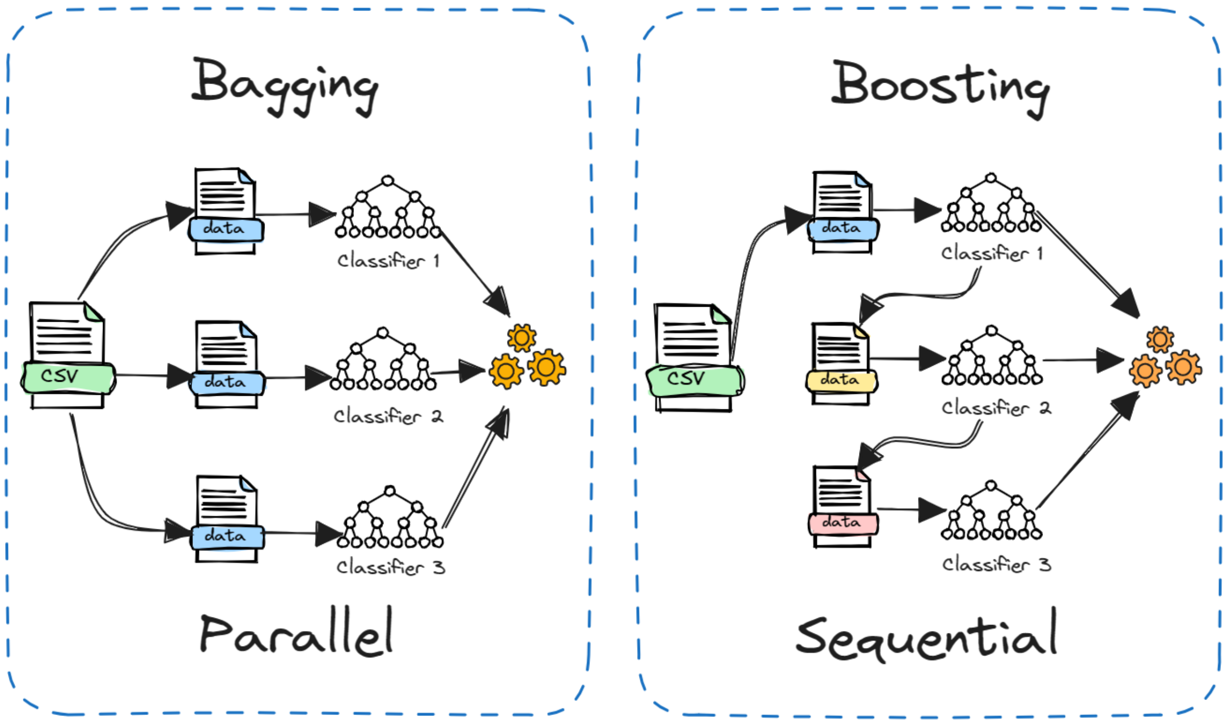
Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time.



Bagging is used in Random Forest Classifier. Bagging trains individual decision trees on random samples of subsets of the dataset to reduce correlation. A benefit of bagging over boosting is that bagging can be performed in parallel while boosting is a sequential operation.

Bagging

Each model is trained on a random subset of the data sampled with replacement, meaning that the individual data points can be chosen more than once. This random subset is known as a bootstrap sample. By training models on different bootstraps, bagging reduces the variance of the individual models. It also avoids overfitting by exposing the constituent models to different parts of the dataset.


Bagging
csv
data
data
data
Classifier 1
Classifier 2
Classifier 3
Parallel
Boosting
csv
data
data
data
Classifier 1
Classifier 2
Classifier 3
Sequential