Random Forest

DataDNA

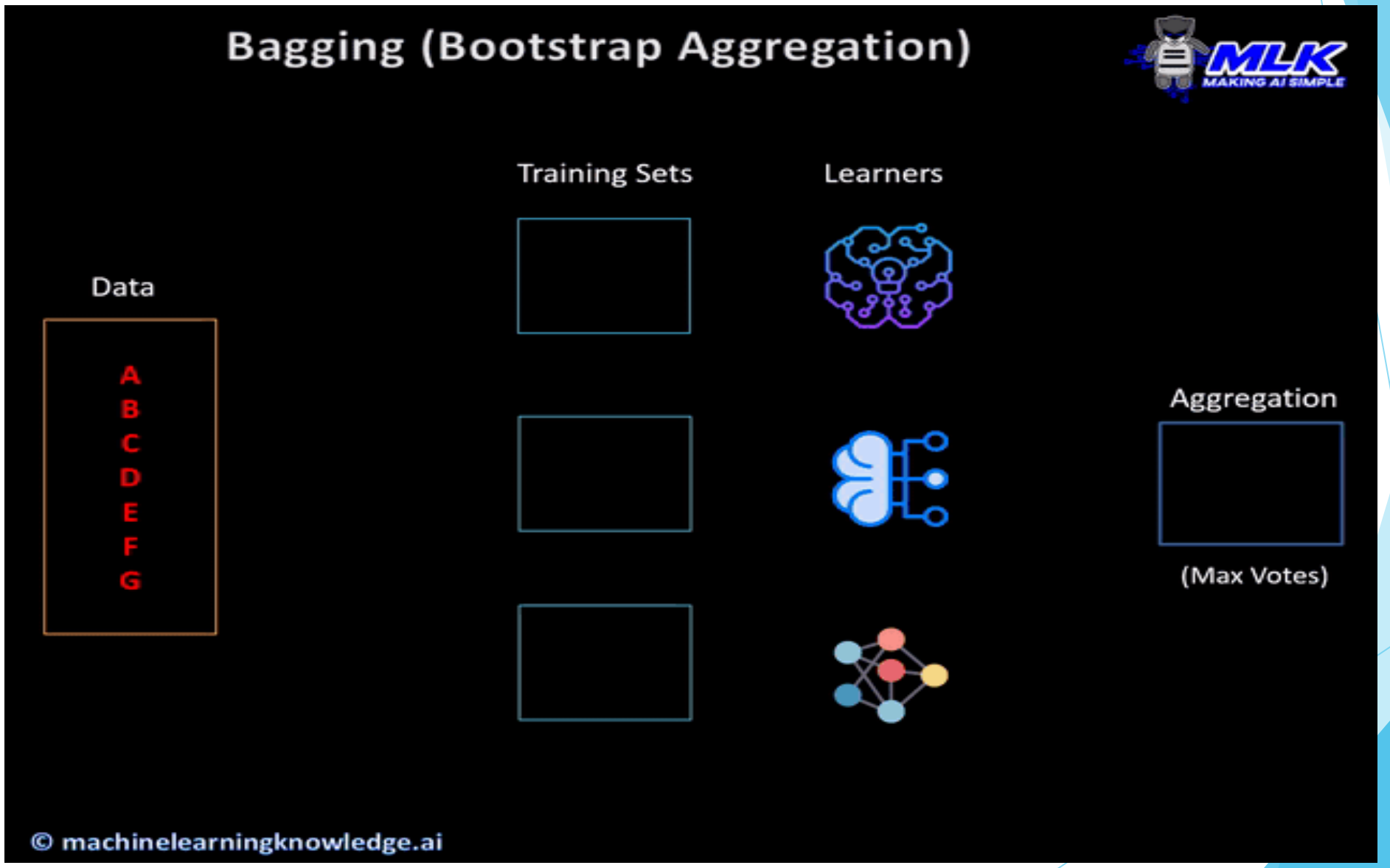
Introduction

- Random forest, consists of many individual decision trees that operate as an ensemble, based on the **divide-and-conquer approach**
- The individual decision trees are generated using an attribute selection indicator such as information gain, gain ratio, and Gini index for each attribute.
- Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.
- The strength of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.
- In a classification problem, each tree votes and the most popular class is chosen as the result. In the case of regression, the average of all the tree outputs is considered as the result.

Bagging Introduction

- **Bagging**, also known as **Bootstrap Aggregation** is the ensemble technique used by random forest.
- Random samples generated from the original data set with replacement known as **row sampling**.
- This step of row sampling with replacement is called **bootstrap**.
- Model will be trained on individual bootstrapped datasets. The final output of the system will be based on most votes after combining results of all individual models. This step of combining all the results of the models and generate output based on the majority voting is called **Aggregation**

Bagging Intuition



DataDNA

Assumptions while creating Random Forest

Some of the assumptions we make while using Decision tree:

- At the beginning, the whole training set is considered as the root.
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- Records are distributed recursively based on attribute values.
- Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

DataDNA

Features of Random Forests

- It runs efficiently on large data bases.
- It can handle lots of input variables without variable deletion.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing error in class population unbalanced data sets.
- Generated forests can be saved for future use on other data.
- Unlike linear models, It maps non-linear relationships quite well.
- It is adaptable at solving any kind of problem at hand (classification or regression).
- It provides the relative feature importance, which helps in selecting the most contributing features for the classifier.
- **Parallelization -** Each tree is created independently out of different data and attributes. This means that we can make full use of the CPU to build random forests.

DataDNA

Downsides

- RF is slow in generating the predictions because it has multiple decision trees through which the input data passed, and each decision tree will make its prediction and subsequently voting will have to performed on top of the predictions from each individual trees. This whole process is time consuming.

- Interpretation is difficult as it contains lot of trees. Not straightforward like decision trees. Hence, Decision tree has a provision of getting converted to trees easily whereas in RF, because of various decision trees in it, deriving rules out of it is cumbersome.

Common terms used with Random Forest:

1.Root Node: It represents entire population or sample and this further gets divided into two or more homogeneous sets.

2.Splitting: It is a process of dividing a node into two or more sub-nodes.

3.Decision Node: When a sub-node splits into further sub-nodes, then it is called decision node.

4.Leaf/ Terminal Node: Nodes do not split is called Leaf or Terminal node.

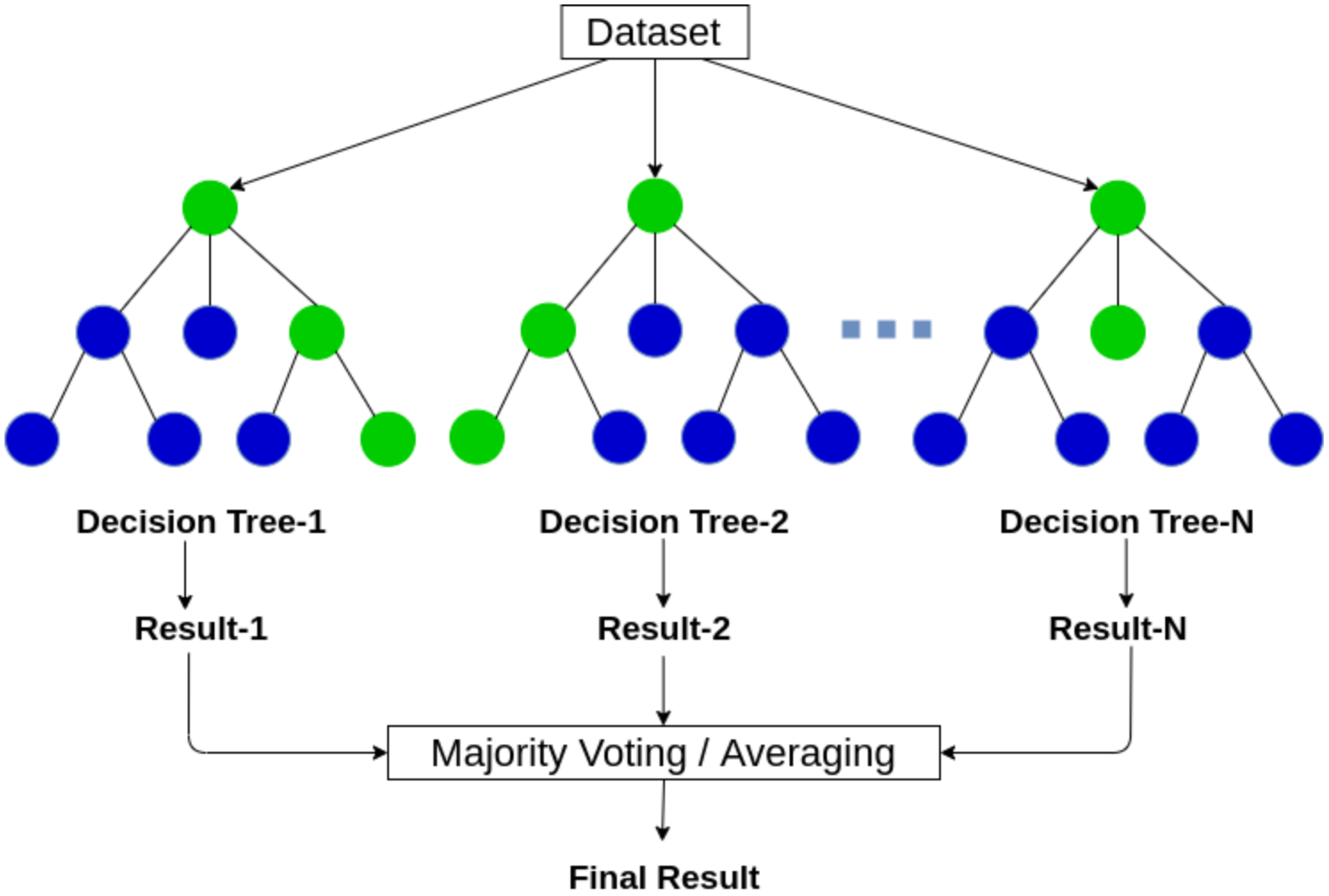
5.Pruning: When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.

6.Branch / Sub-Tree: A sub section of entire tree is called branch or sub-tree.

7.Parent and Child Node: A node, which is divided into sub-nodes is called parent node of sub-nodes whereas sub-nodes are the child of parent node.

DataDNA

Sample Random Forest



DataDNA

How a Random Forest Works

1. Randomly select "**K**" features from total "**m**" features where **k << m**
2. Among the "**K**" features, calculate the node "**d**" using the best split point
3. Split the node into **child nodes** using the **best split**
4. Repeat the **1 to 3** steps until "l" number of nodes has been reached
5. Build forest by repeating steps **1 to 4** for "n" number times to create **"n" number of trees**

Calculating Entropy for the root node

$$E = -\left(P(\checkmark) * \log_2(P(\checkmark)) + P(\times) * \log_2(P(\times))\right)$$

Probability formula:

$$P(\checkmark) = \frac{\text{count of } \checkmark}{\text{total examples}}$$

$P(\checkmark) = 2/4 = 0.5$

$P(\times) = 2/4 = 0.5$

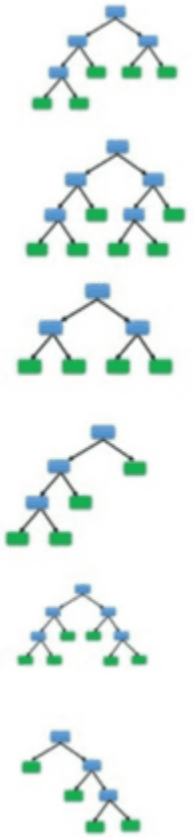
Plugging these values in the formula we get:

$$E = -\left(0.5 * \log_2(0.5) + 0.5 * \log_2(0.5)\right)$$

$$E = 1$$

DataDNA

Sample Random Forest



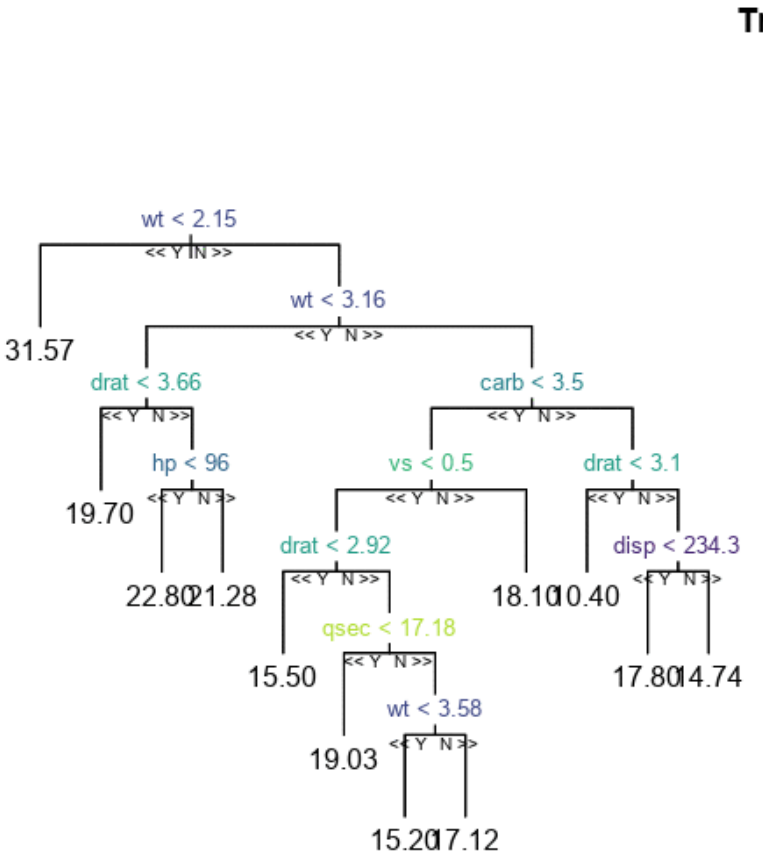
Random Forest in Action!!!

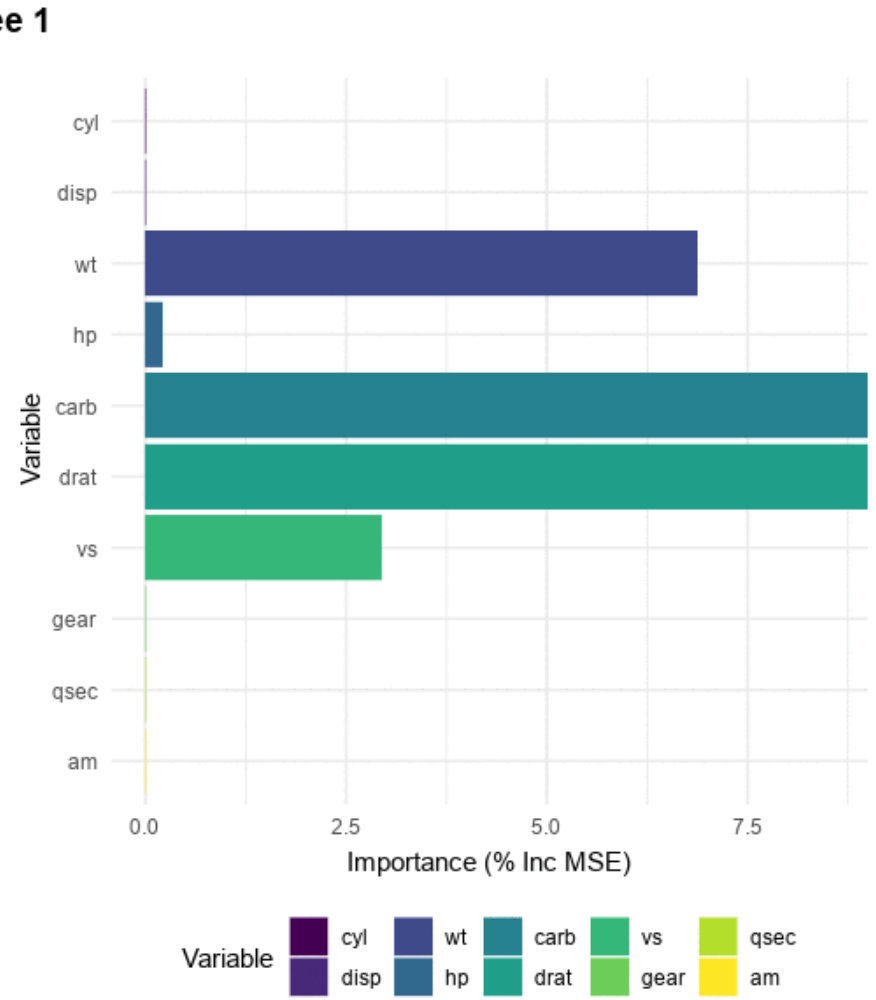
Intuition behind Variable Importance

1) The default method to compute variable importance is the ***mean decrease in impurity*** (or *Gini importance*) mechanism

2) At each split in each tree, the improvement in the split-criterion is the importance measure attributed to the splitting variable and is accumulated over all the trees in the forest separately for each variable.

Intuition behind Variable Importance





Hyper parameter Tuning

criterion*{"gini", "entropy"}, default="gini" (ClassificationType)*
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

criterion*{"mse", "mae"}, default="mse" (Regressor Type)*
The function to measure the quality of a split. Supported criteria are "mse" for the mean squared error, which is equal to variance reduction as feature selection criterion, and "mae" for the mean absolute error.

splitter*{"best", "random"}, default="best"*
The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth *int*, *default=None*
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split *int or float*, *default=2*
The minimum number of samples required to split an internal node.

min_samples_leaf int or float, default=1
The minimum number of samples required to be at a leaf node after splitting. 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. This may have the effect of smoothing the model, especially in regression.

Hyper Parameter Tuning

max_features int, float or {“auto”, “sqrt”, “log2”}, default=None
The number of features to consider when looking for the best split:

max_leaf_nodes int, default=None

Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity.
If None then unlimited number of leaf nodes.

min_impurity_decrease float, default=0.0

A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

max_samples int or float, default=None
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.
- If int, then draw max_samples samples.
- If float, then draw max_samples * X.shape[0] samples. Thus, max_samples should be in the interval (0, 1).

Bootstrap *bool, default=True*
Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

oob_score *bool, default=False*
whether to use out-of-bag samples to estimate the R^2 on unseen data.

Model Evaluation Metrics

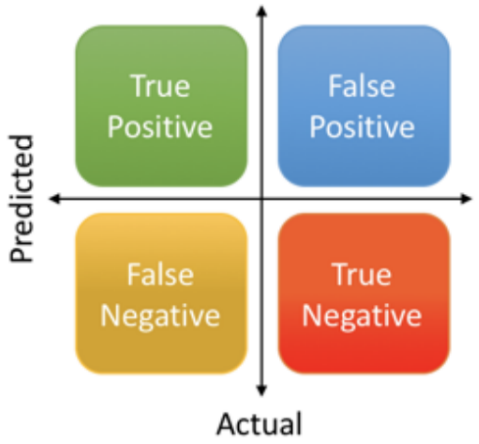
The various metrics used to evaluate the results of the prediction are :

1.Accuracy
2.Precision
3.Recall
4.Area Under ROC Curve
5.MAPE/ MAD/ RMSE (For Regressors)

$$\textbf{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

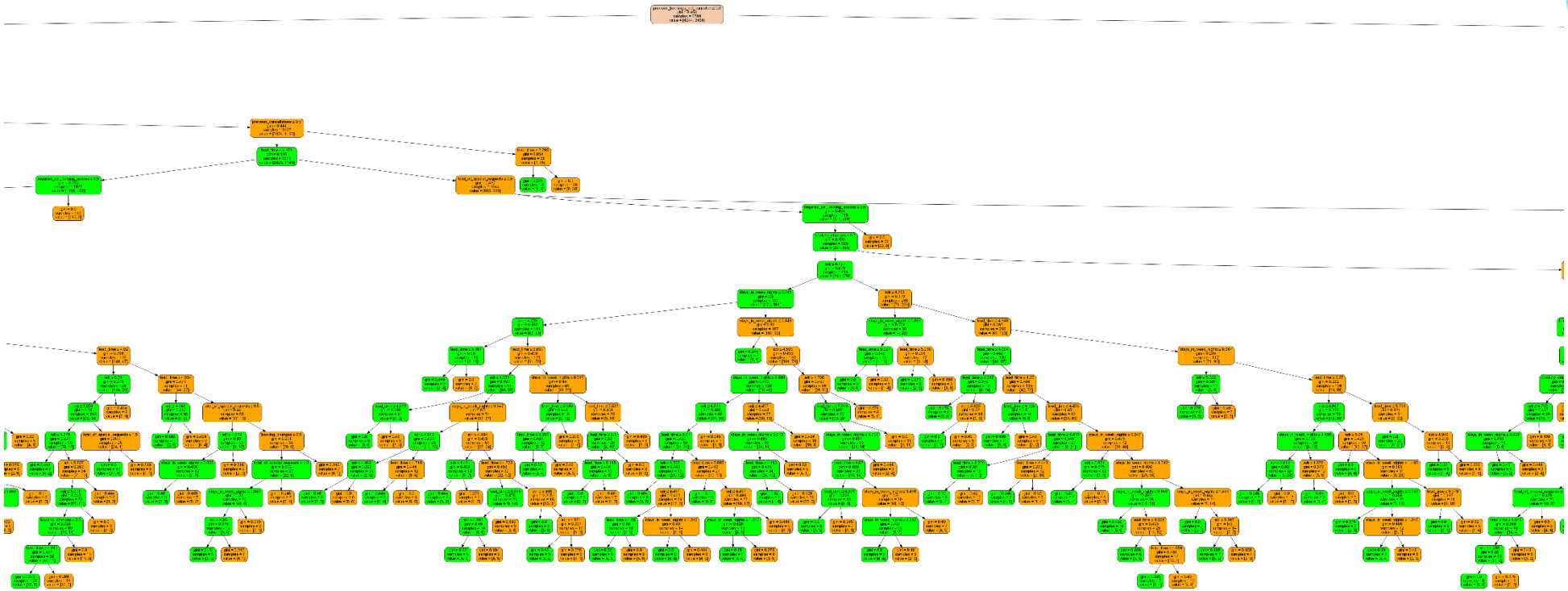
$$\textbf{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

$$\textbf{Accuracy} = \frac{\text{True Positive + True Negative}}{\text{Total}}$$



DataDNA

Model Inference



Industry Applications

Banking Industry

- **Credit Card Fraud Detection**
- **Customer Segmentation**
- **Predicting Loan Defaults on LendingClub.com**

Healthcare and Medicine

- **Cardiovascular Disease Prediction**
- **Diabetes Prediction**
- **Breast Cancer Prediction**

Stock Market

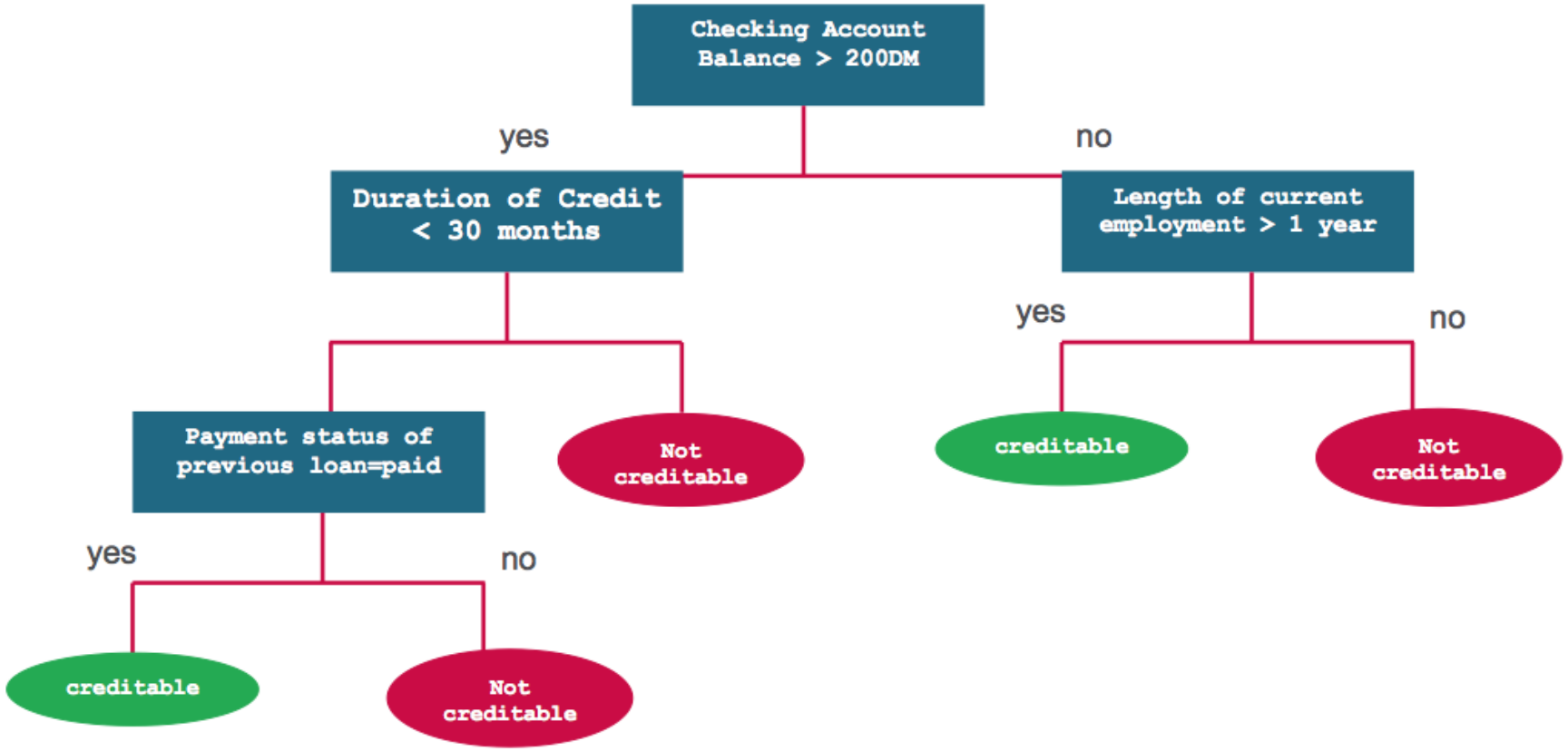
- **Stock Market Prediction**
- **Stock Market Sentiment Analysis**
- **Bitcoin Price Detection**

E-Commerce

- **Product Recommendation**
- **Price Optimization**
- **Search Ranking**

DataDNA

Applications : Industry



References

https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/