Introduction to Machine Learning in Geosciences GEO371T/GEO398D.1

Ensemble-based Methods

Mrinal K. Sen
Geolsciences
UT Austin

October 31, 2023

Tree

Content

- Introduction
- Classification and Regression Trees (CART)
- Bagging (Bootstrap aggregation) - a general variance reduction technique
- Random Forests
- Boosting

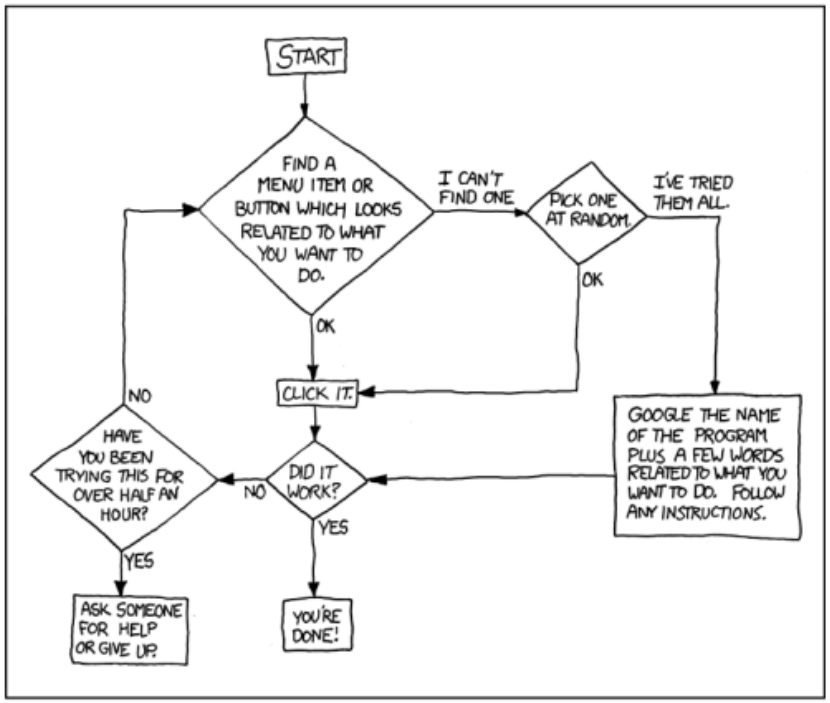
Introduction

- Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model.
- We will start with an example, and utilize Decision Trees to outline the definition and practicality of Ensemble Methods
- First, high dimensional data impose computational challenges.
- Moreover, in some situations high dimensionality might lead to poor generalization abilities of the learning algorithm.
- Finally, dimensionality reduction can be used for interpretability of the data, for finding meaningful structure of the data, and for illustration purposes.

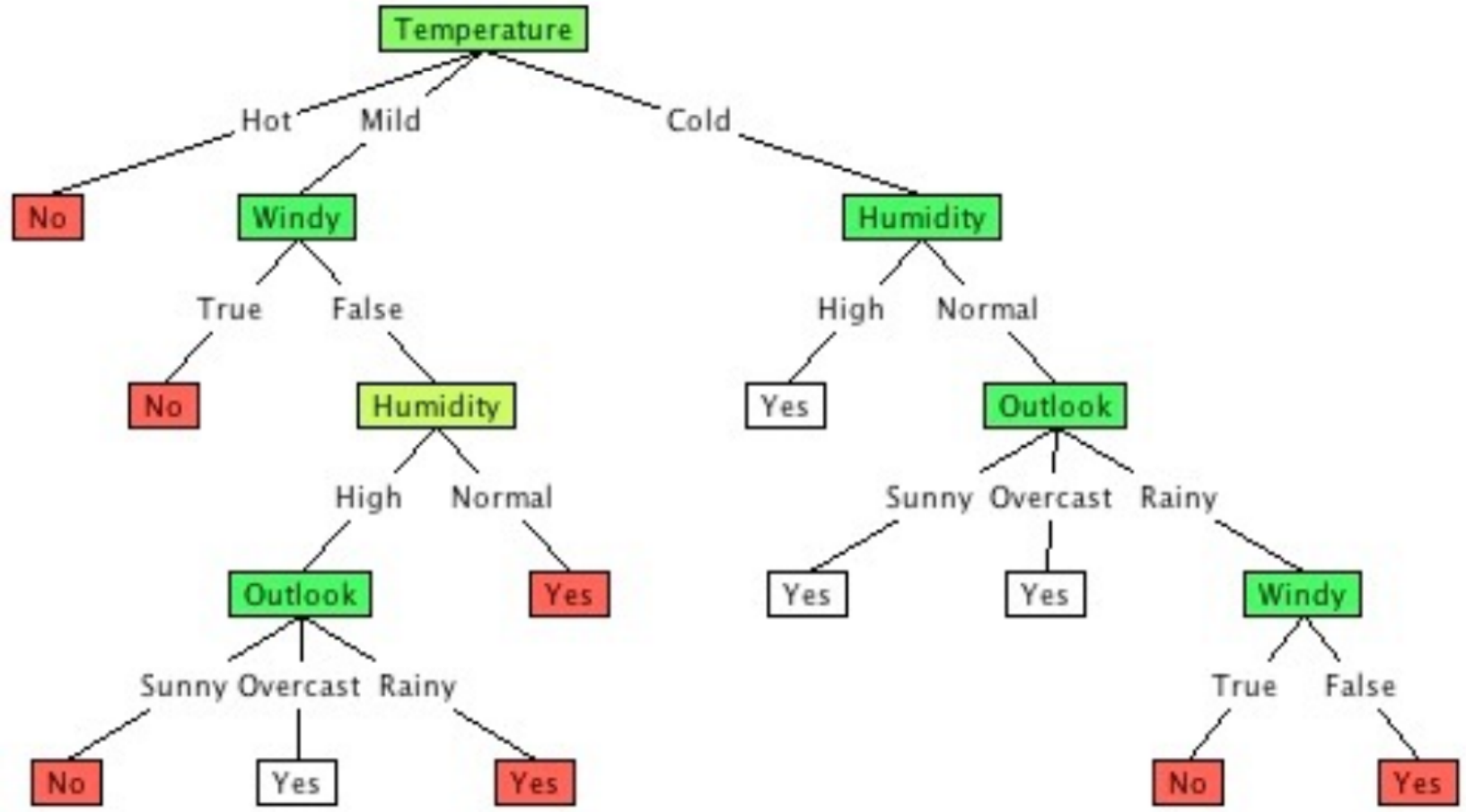
MOTIVATION

- How do we make decisions?
  - Consider a variety of factors
  - Follow a logical path of checks
- Should I eat at this restaurant?
  - If there is no wait
    - **YES**
  - If there is short wait and I am hungry
    - **YES**
  - Else
    - **NO**

Decision Flowchart



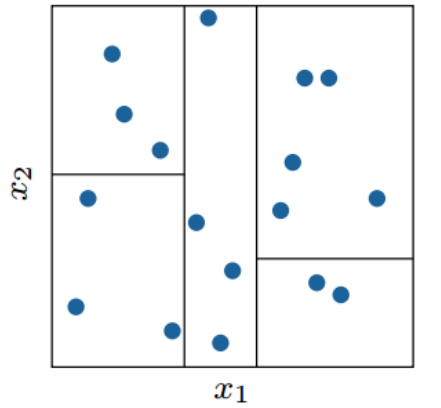
Introduction



Basic idea behind tree-based Methods

In both regression and classification settings we seek a function $\widehat{y}(\mathbf{x})$ which maps the input $\mathbf{x}$ into a prediction.

One **flexible** way of designing this function is to partition the input space into disjoint regions and fit a simple model in each region.



- **Classification:** Majority vote within the region.
- **Regression:** Mean of training data within the region.

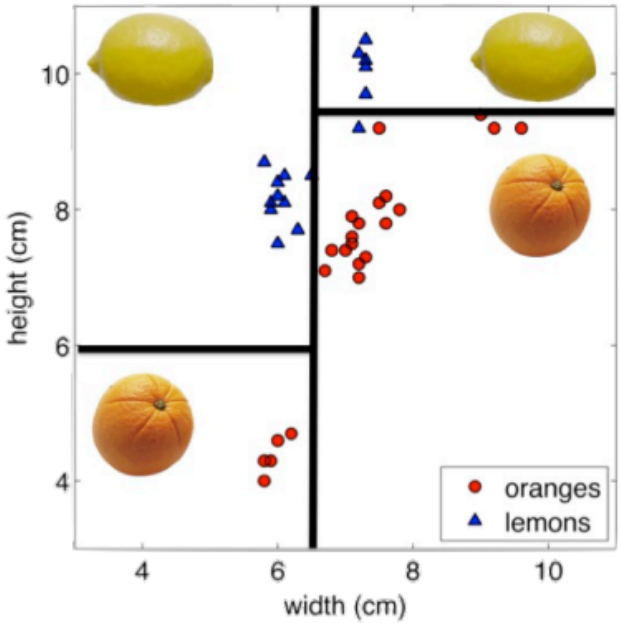
niklas.wahlstrom@it.uu.se

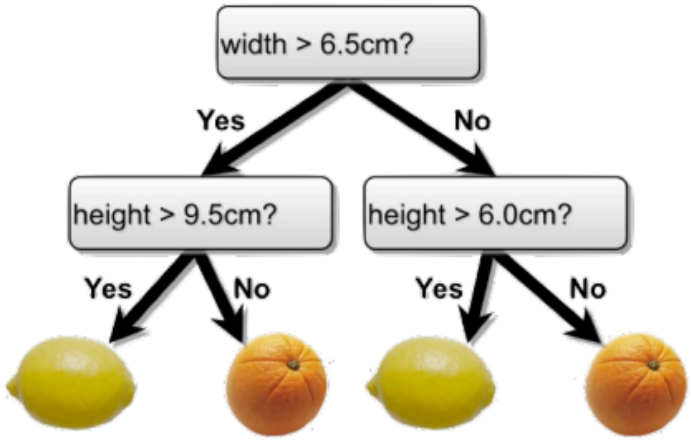
Introduction

A ***decision tree model*** is an interpretable model in which the final output is based on a series of comparisons of the values of predictors against threshold values.

Graphically, decision trees can be represented by a flow chart.

Geometrically, the model partitions the feature space wherein each region is assigned a response variable value based on the training points contained in the region.



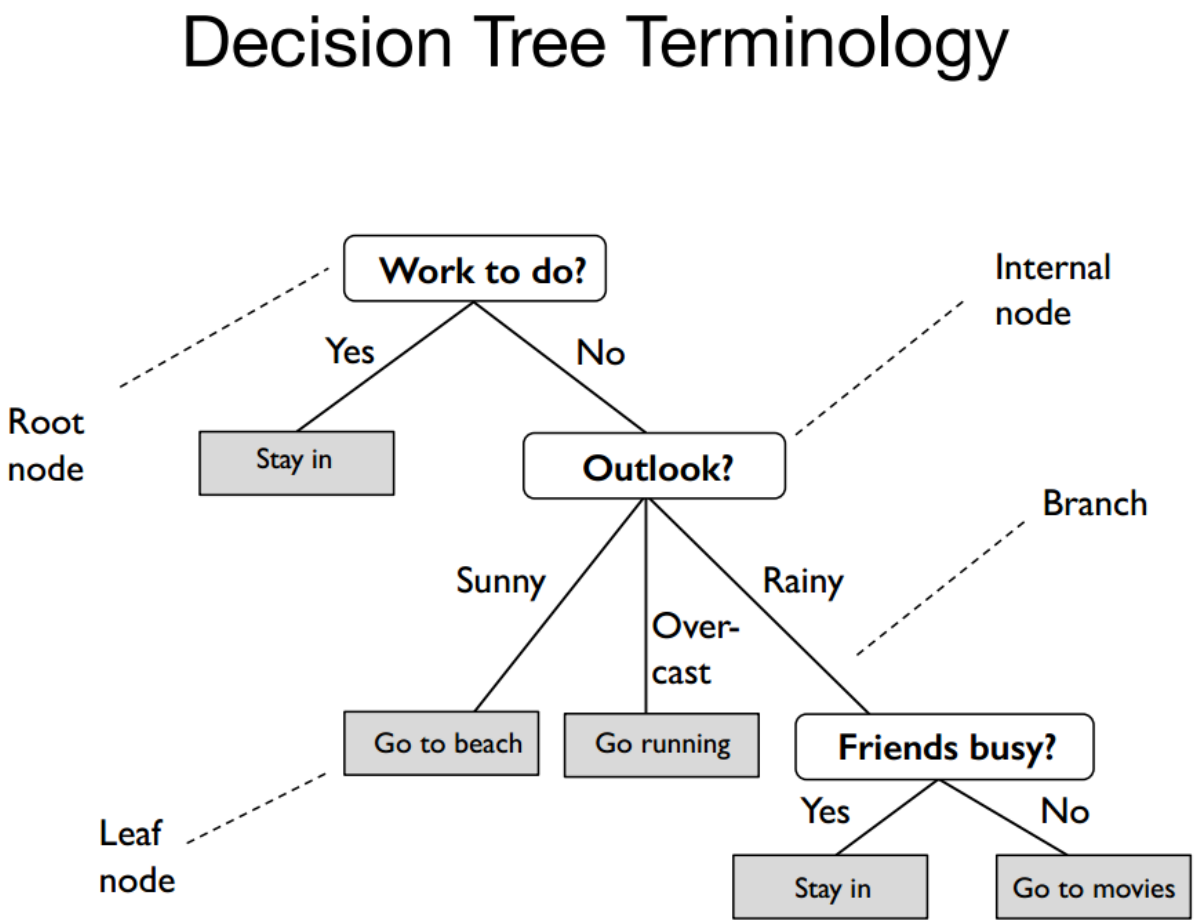


Introduction

- An iterative, top-down construction of the hypothesis; picture a hierarchy of decision, forking the dataset into subspaces.
- Can represent any Boolean (binary) function, and the hypothesis space being searched is the entire space of Boolean functions; however, we need to keep in mind that a critical challenge in machine learning is whether an algorithm can learn/find the "right" function or a good approximation.
- Considering only binary (or Boolean) features, at each node, there 2^m potential splits to be evaluated given m features.
- Hypothesis space is searched greedily (i.e., decision tree algorithms perform a greedy search over all possible trees)

http://stat.wisc.edu/ sraschka/teaching/stat479-fs2018/

Introduction



http://stat.wisc.edu/ sraschka/teaching/stat479-fs2018/

Terminology

- **Root node**: no incoming edge, zero or more outgoing edges.
- **Internal node**: one incoming edge, two (or more) outgoing edges.
- **Leaf node**: each leaf node is an assigned a class label (if nodes are pure; otherwise majority vote).
- **Parent and child nodes**: If a node is split, we refer to that given node as the parent node, and the resulting nodes are called child nodes, respectively.

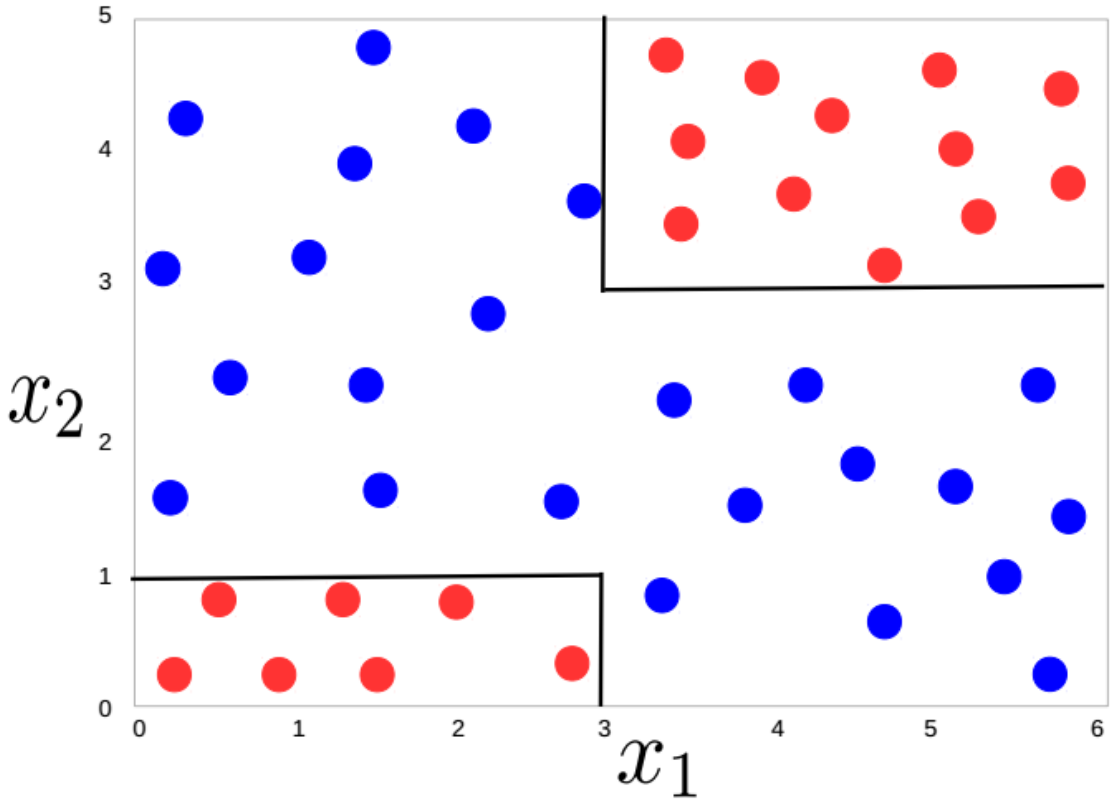
http://stat.wisc.edu/ sraschka/teaching/stat479-fs2018/

Structure

- **Nodes**: Tests for variables
- **Branches**: Results of tests
- **Leaves**: Classification

A classification Problem

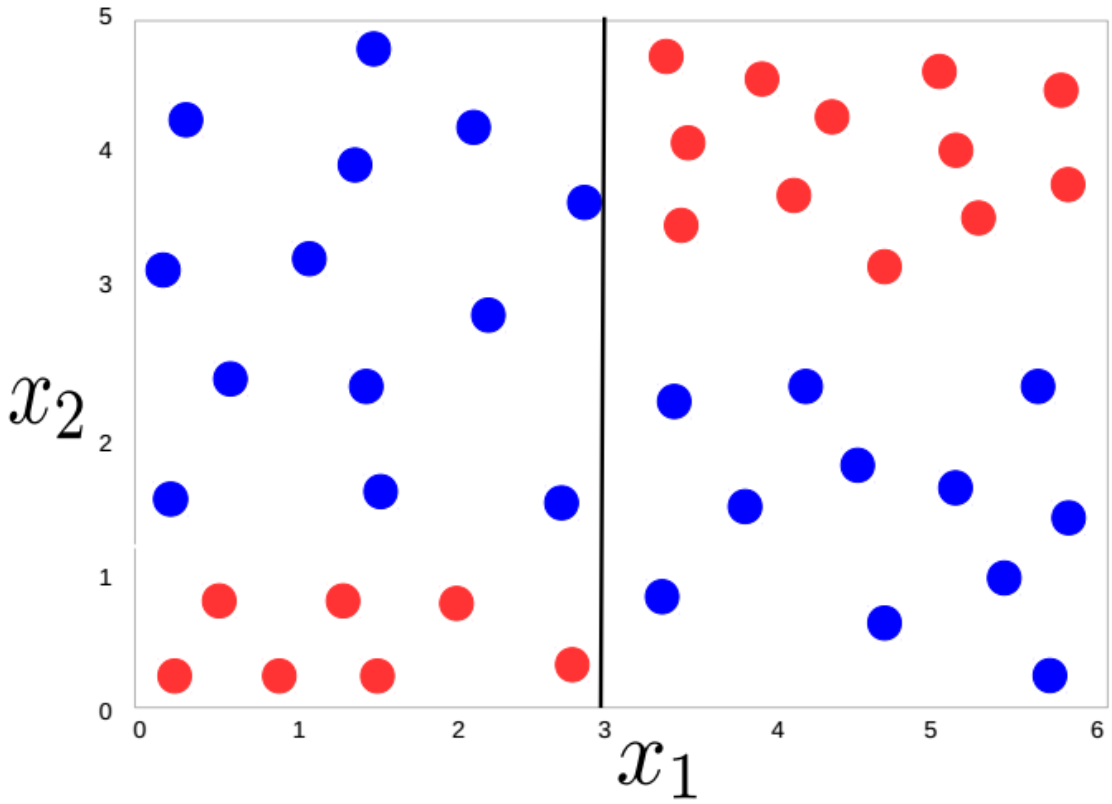
The "expected" decision boundary given this training data. Let's learn this!



https://www.cse.iitk.ac.in/users/piyush/courses/ml_autumn18/index.html

A classification Problem

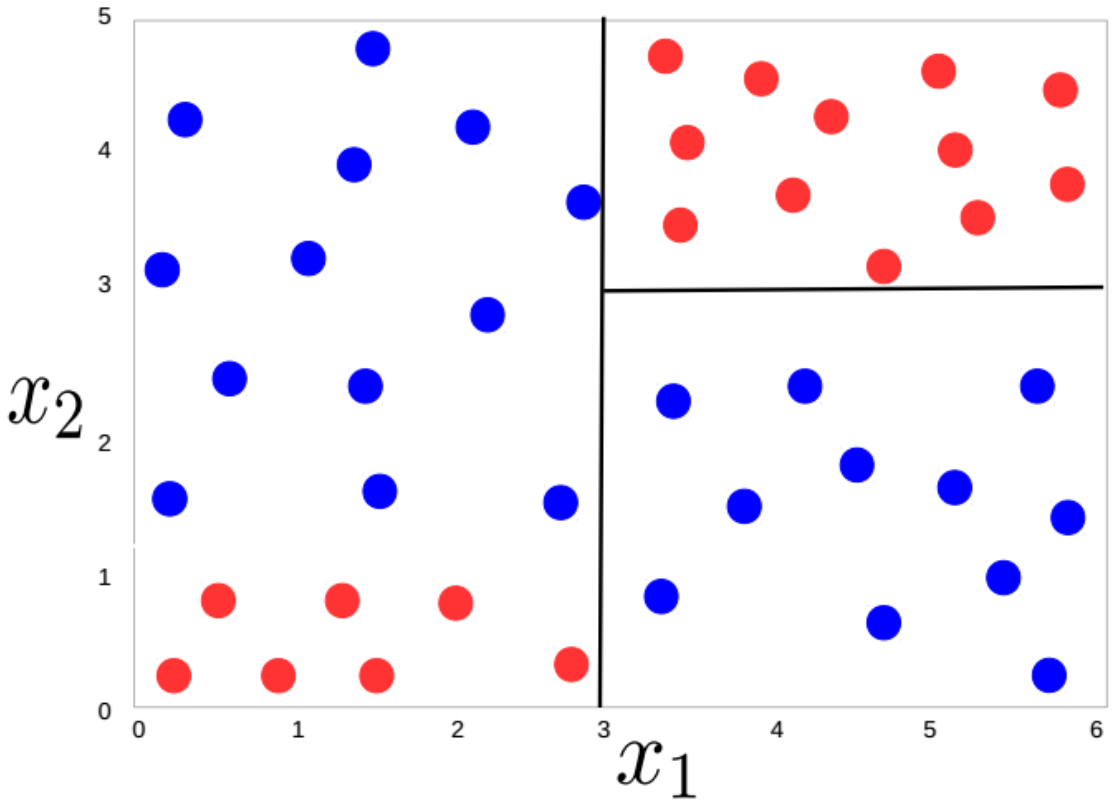
Is x_1 (feature 1) greater than 3 ?



https://www.cse.iitk.ac.in/users/piyush/courses/ml_autumn8/index.html

A classification Problem

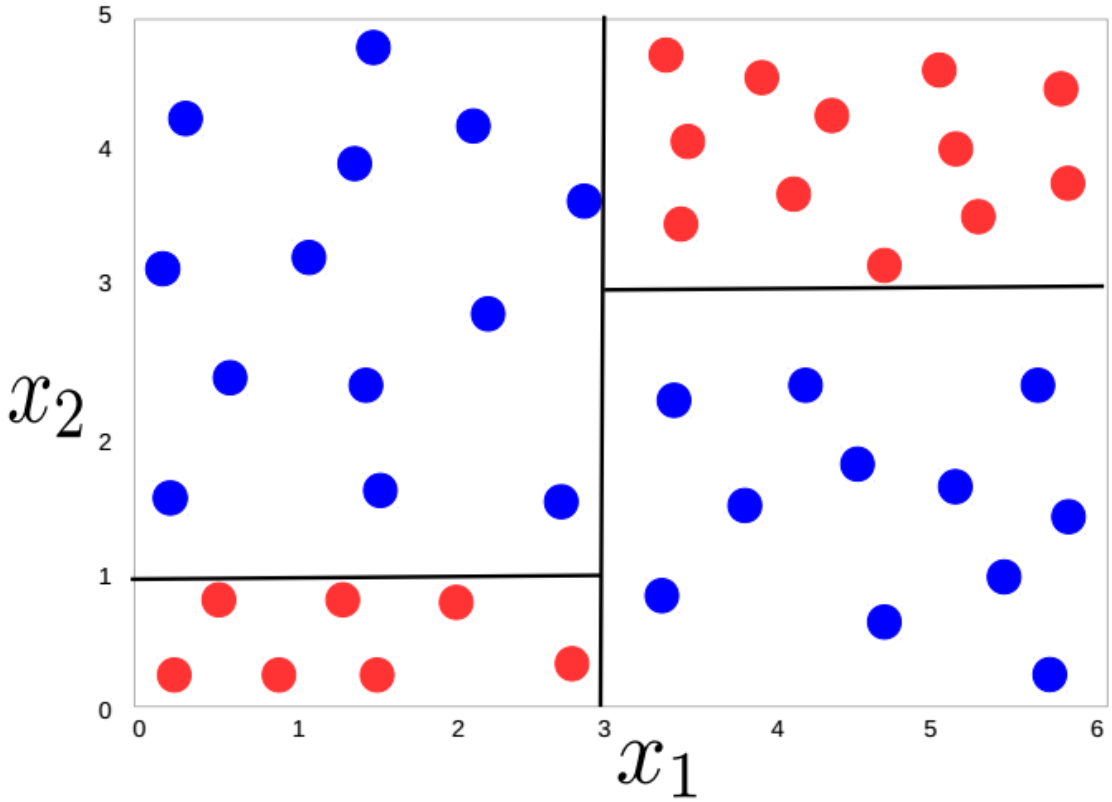
Given $x_1 > 3$, is feature 2 (x_2) greater than 3?



https://www.cse.iitk.ac.in/users/piyush/courses/ml_autumn18/index.html

A classification Problem

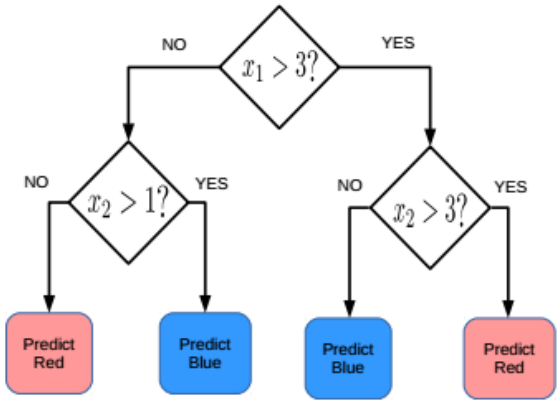
Given $x_1 < 3$, is feature 2 (x_2) greater than 1?

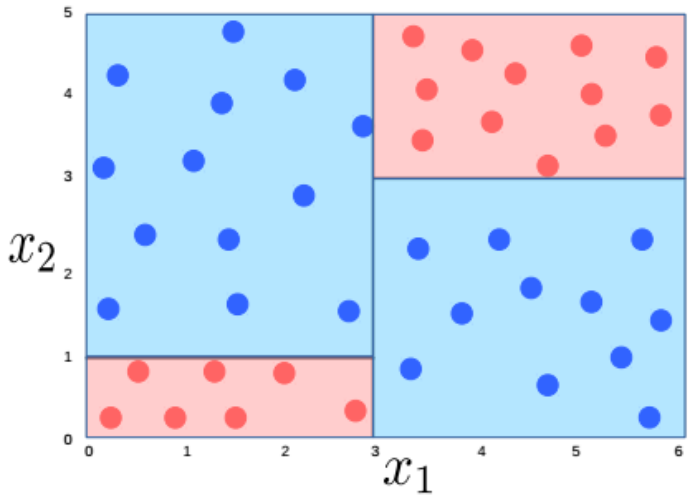


https://www.cse.iitk.ac.in/users/piyush/courses/ml_autumn18/index.html

A classification Problem

- A Decision Tree (DT) consisting of a set of rules **learned** from training data



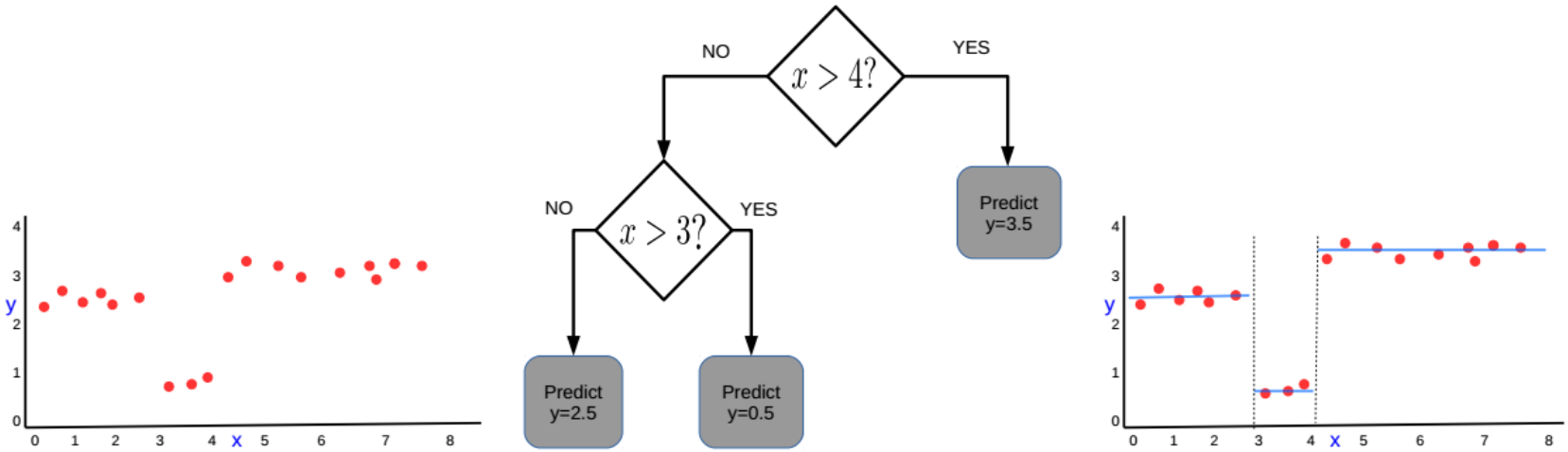


- These rules perform a recursive partitioning of the training data into "homogeneous"
  - Homogeneous means that the outputs are same/similar for all inputs in that region
- Given a new test input, we can use the DT to predict its label
- A key benefit of DT: Prediction at test time is very fast (just testing a few conditions

https://www.cse.iitk.ac.in/users/piyush/courses/ml_autumn18/index.html

A Regression Problem

Decision Trees can also be used for regression problems

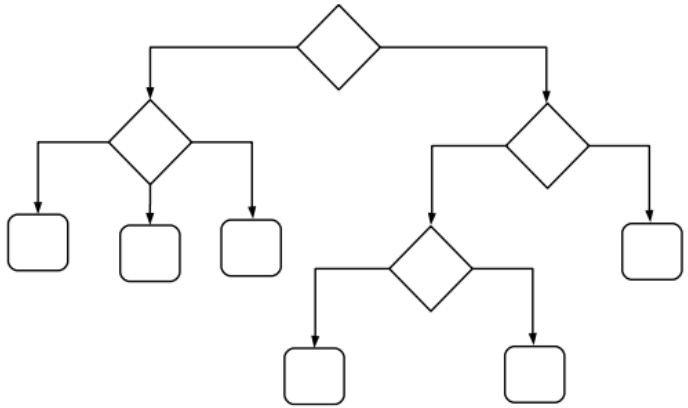


Here too, the DT partitions the training data into homogeneous regions (inputs with similar outputs)

https://www.cse.iitk.ac.in/users/piyush/courses/ml_autumn18/index.html

Shape and Size of DT

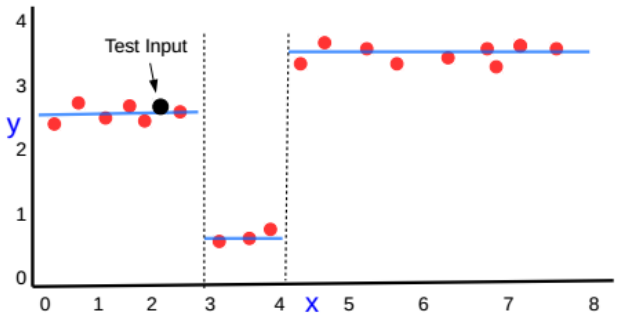
- What should be the size/shape of the DT?
  - Number of internal and leaf nodes
  - Branching factor of internal nodes
  - Depth of the tree

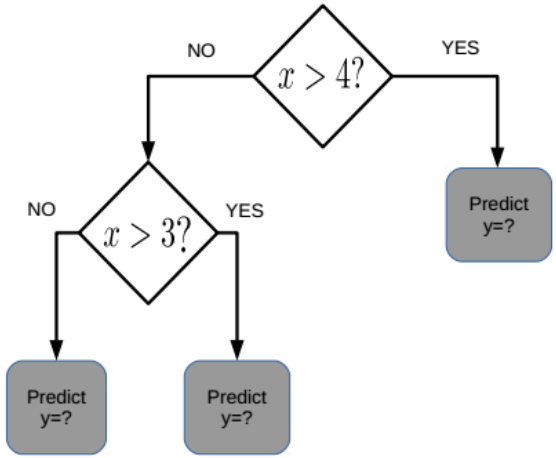


https://www.cse.iitk.ac.in/users/piyush/courses/ml$_a$*utumn*18/*index*.*html*

Leaf Nodes

- What to do at each leaf node (the goal: make predictions)? Some options:
  - Make a constant prediction (majority/average) for every test input reaching that leaf node?
  - Use a nearest neighbors based prediction using all training inputs on that leaf node?



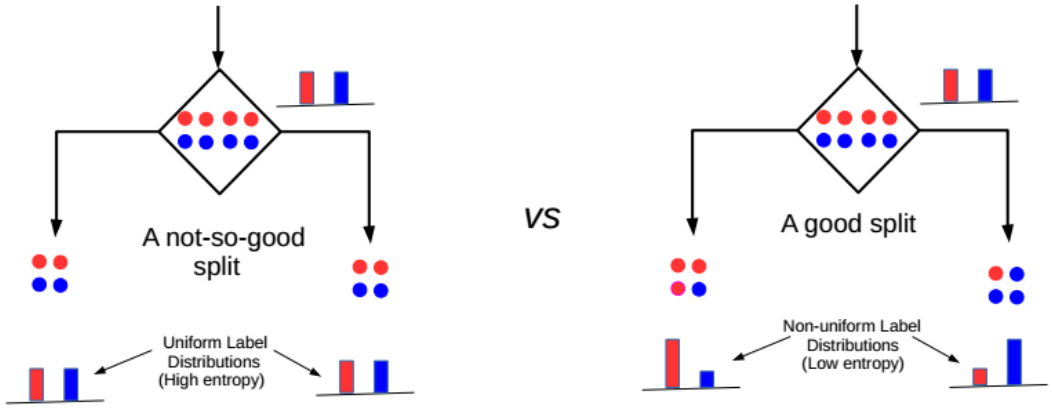


  - (Less common) Predict using an ML model learned using training inputs that belong to that leaf node?
- Constant prediction is the fastest at test time (and gives a piece-wise constant prediction rule)

https://www.cse.iitk.ac.in/users/piyush/courses/ml$_a$*utumn*18/*index.html*

Internal Nodes

- How to split at each internal node (the goal: split the training data received at that node)?
- No matter how we split, the goal should be to have splits that result in groups as "pure" as poss

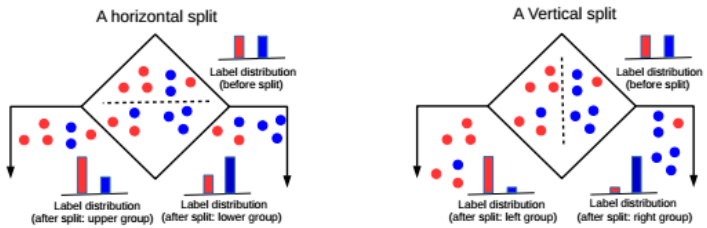


- For classification problems, entropy of the label distribution is a measure of purity
  - Low entropy $\Rightarrow$ high purity (less uniform label distribution)
  - Splits that give the largest reduction (before split vs after split) in entropy are preferred (this reduction is also known as "information gain")
- For regression, entropy doesn't make sense (outputs are real-valued). Typically variance is used.

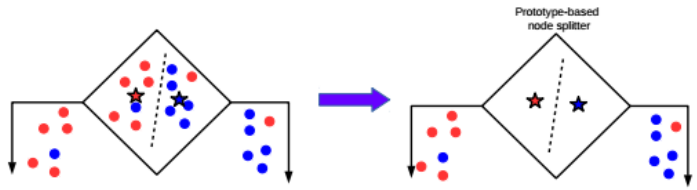
https://www.cse.iitk.ac.in/users/piyush/courses/ml_autumn18/index.html

Internal Nodes

- Note that splitting at internal node itself is like a classification problem
  - Data received at the internal node has to be routed along its outgoing branches
- Some common techniques for splitting an internal node
  - Splitting by testing a single feature (simplest/fastest; used in ID3, C4.5 DT algos). For example:



  - Splitting using a classifier learned using data on that node. For example, prototype based classifier



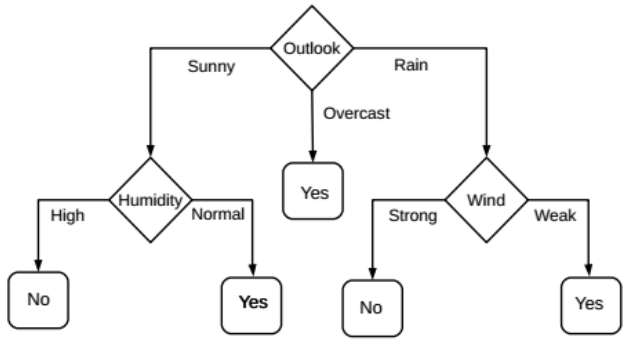
- The same splitting rule will be applied to route a test input that reaches this internal node

https://www.cse.iitk.ac.in/users/piyush/courses/ml$_a$*utumn*18/*index.html*

Decision Tree Construction

- As an illustration, let's look at one way of constructing a decision tree for some given data
- We will use the entropy/information-gain based splitting criterion for this illustration

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



- Question: Why does it make more sense to test the feature "outlook" first?
- Answer: Of all the 4 features, it's most informative (highest information gain as the root node)

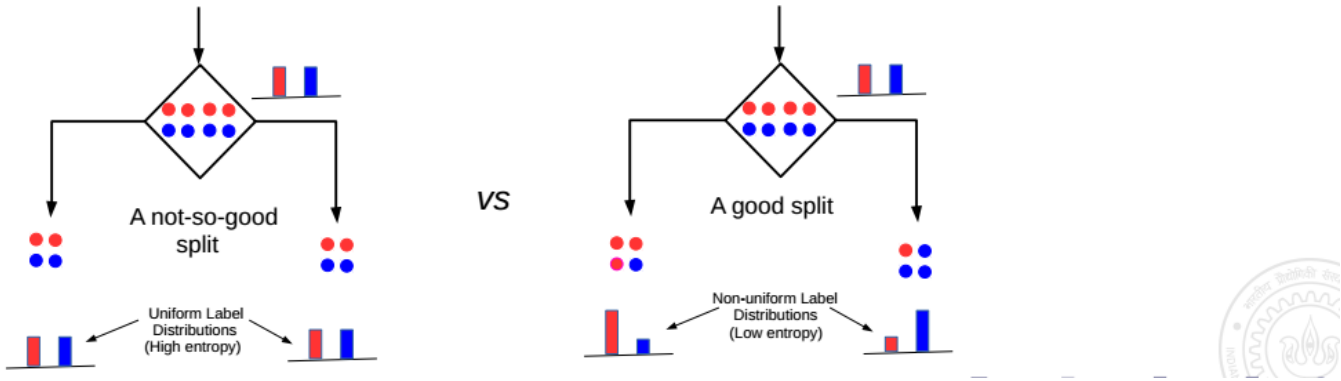
https://www.cse.iitk.ac.in/users/piyush/courses/ml$_a$*utumn*18/*index.html*

Entropy and Information Gain

- Consider a set S of inputs with a total C classes, p_c = fraction of inputs from class/label c
- Entropy of the set S: $H(S) = -\sum_{c \in C} p_c \log_2 p_c$
- The difference in the entropy before and after the split is called information gain (IG)
- For one group S being split into two smaller groups S_1 and S_2, we can calculate the IG as follows

$$IG = H(S) - \frac{|S_1|}{|S|} H(S_1) - \frac{|S_2|}{|S|} H(S_2)$$

- For DT construction, entropy/IG gives us a criterion to select the best split for an internal node



https://www.cse.iitk.ac.in/users/piyush/courses/ml_autumn18/index.html

Entropy and Information Gain

- Let's look at IG based DT construction for the Tennis example
- Let's begin with the root node of the DT and compute IG of each feature
- Consider feature "wind" $\in$ {weak,strong} and its IG at root

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no

- Root node: $S = [9+, 5-]$ (all training data: 9 play, 5 no-play)
- Entropy: $H(S) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) = 0.94$
- $S_{weak} = [6+, 2-] \Longrightarrow H(S_{weak}) = 0.811$, $S_{strong} = [3+, 3-] \Longrightarrow H(S_{strong}) = 1$

$$\begin{aligned} IG(S, \text{wind}) &= H(S) - \frac{|S_{weak}|}{|S|} H(S_{weak}) - \frac{|S_{strong}|}{|S|} H(S_{strong}) \\ &= 0.94 - 8/14 * 0.811 - 6/14 * 1 \\ &= 0.048 \end{aligned}$$

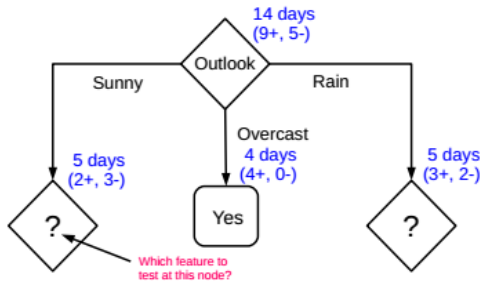
Likewise, $IG(S, \text{outlook}) = 0.246$, $IG(S, \text{humidity}) = 0.151$, $IG(S, \text{temperature}) = 0.029 \Rightarrow$ outlook chosen

https://www.cse.iitk.ac.in/users/piyush/courses/ml_autumn18/index.html

Growing the tree

- Having decided which feature to test at the root, let's grow the tree
- How to decide which feature to test at the next level (level 2) ?
- **Rule:** Iterate - for each child node, select the feature with the highest IG

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no

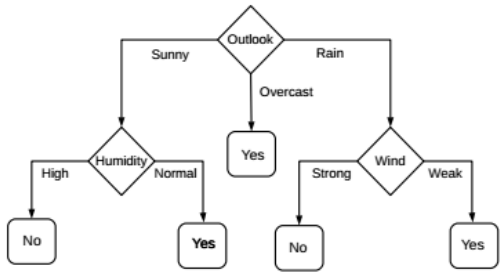


- Proceeding as before, for level 2, left node, we can verify that
  - $IG(S, \text{temperature}) = 0.570$, $IG(S, \text{humidity}) = 0.970$, $IG(S, \text{wind}) = 0.019$ (thus humidity chosen)
- No need to expand the middle node (already "pure" - all yes)
- Can also verify that wind has the largest IG for the right node
- Note: If a feature has already been tested along a path earlier, we don't consider it again

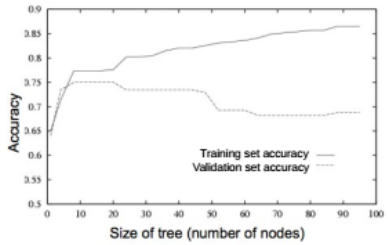
https://www.cse.iitk.ac.in/users/piyush/courses/ml_autumn18/index.html

Stopping the tree

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



- Stop expanding a node further when
  - It consist of examples all having the same label (the node becomes "pure")
  - We run out of features to test along the path to that node
  - The DT starts to overfit (can be checked by monitoring the validation set accuracy)



https://www.cse.iitk.ac.in/users/piyush/courses/ml$_a$*utumn*18/*index.html*

Avoiding Overfitting: Decision tree pruning

- Desired: a DT that is not too big in size, yet fits the training data reasonably
- Note: An example of a very simple DT is "decision-stump"
  - A decision-stump only tests the value of a single feature
  - Not very powerful in itself but often used in large ensembles of decision stumps
- Mainly two approaches to prune a complex DT
  - Prune while building the tree (**stopping early**)
  - Prune after building the tree (**post-pruning**)
- Criteria for judging which nodes could potentially be pruned
  - Use a validation set (separate from the training set)
    - Prune each possible node that doesn't hurt the accuracy on the validation set
    - Greedily remove the node that improves the validation accuracy the most
    - Stop when the validation set accuracy starts worsening
  - Use model selection methods, such as Minimum Description Length (MDL); more on this later

https://www.cse.iitk.ac.in/users/piyush/courses/ml_autumn18/index.html

Comments

- Other alternatives to entropy for judging feature informativeness in DT classification?
  - Gini-index $\sum_{c=1}^{C} p_c(1 - p_c)$ is another popular choice
- For DT regression (Regression Trees[1]), can split based on the variance in the outputs, instead of using entropy (which doesn't make sense for real-valued inputs)
- Real-valued features (we already saw some examples) can be dealt with using thresholding
  - Need to be careful w.r.t. number of threshold points, how fine each range is, etc.
- More sophisticated decision rules at the internal nodes can also be used (anything that splits the inputs at an internal node into homogeneous groups; e.g., a machine learning classification algo)
- Need to take care handling training or test inputs that have some features missing

https://www.cse.iitk.ac.in/users/piyush/courses/ml_autumn18/index.html

Summary

Some key strengths:

- Simple and each to interpret
- Do not make any assumption about distribution of data
- Easily handle different types of features (real, categorical/nominal, etc.)
- Very fast at test time (just need to check the features, starting the root node and following the DT until you reach a leaf node)
- Multiple DTs can be combined via ensemble methods (e.g., Decision Forest)
  - Each DT can be constructed using a (random) small subset of features

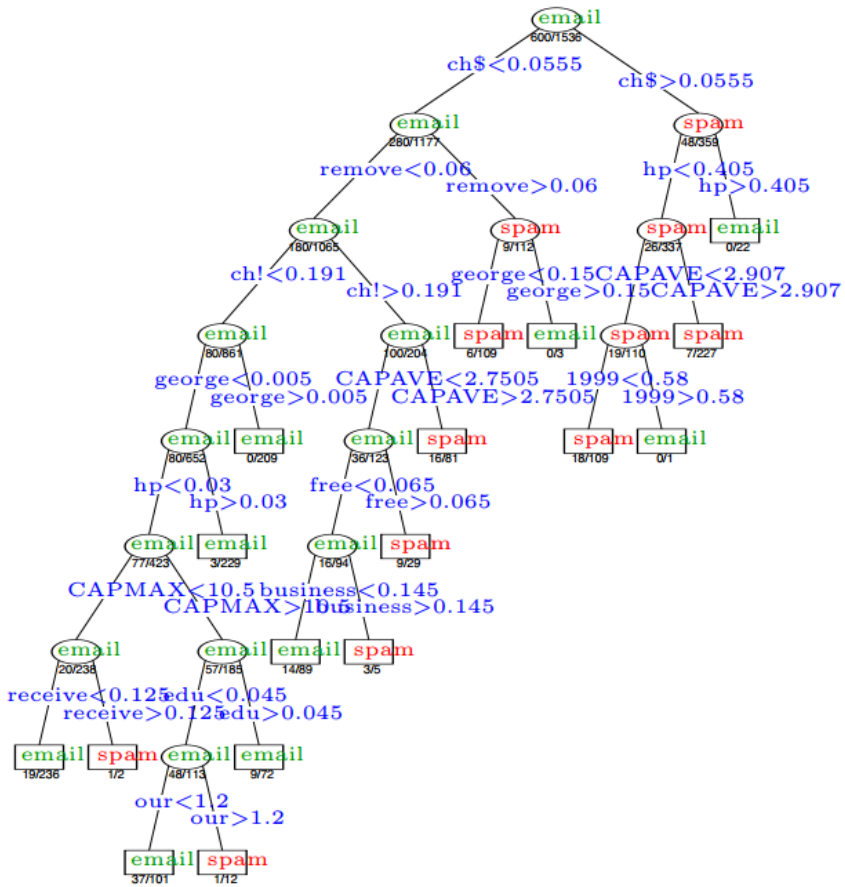
Some key weaknesses:

- Learning the optimal DT is NP-Complete. The existing algorithms are heuristics (e.g., greedy selection of features)
- Can sometimes become very complex unless some pruning is applied

https://www.cse.iitk.ac.in/users/piyush/courses/ml_autumn18/index.html

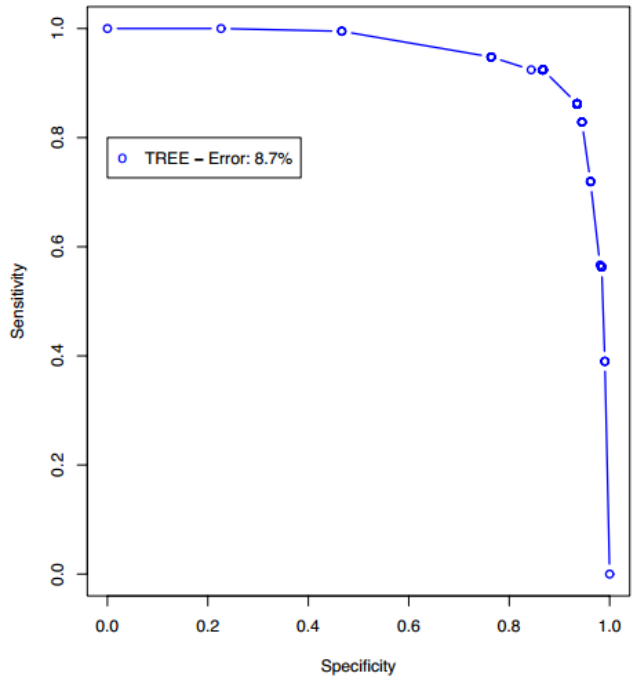
Summary

Decision Tree for Spam Classification



Summary

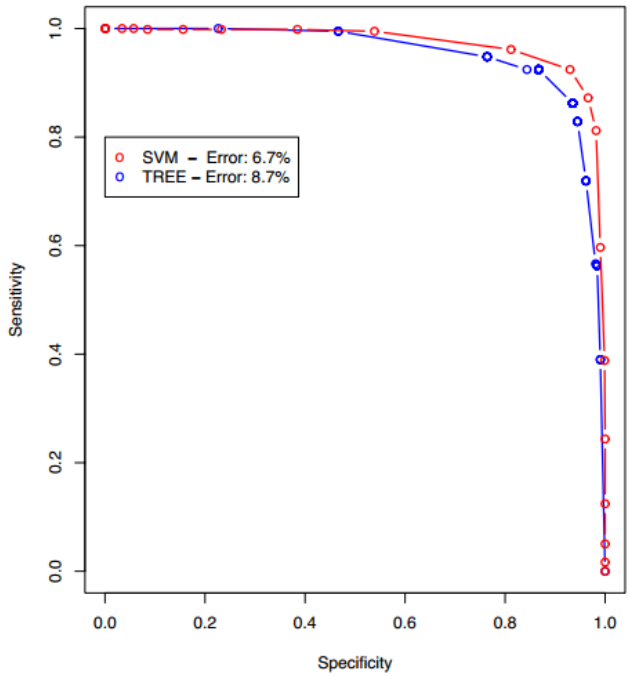
Decision Tree for Spam Classification



- Sensitivity: proportion of true spam identified
- Specificity: proportion of true email identified.

Summary

Decision Tree v.s. SVM

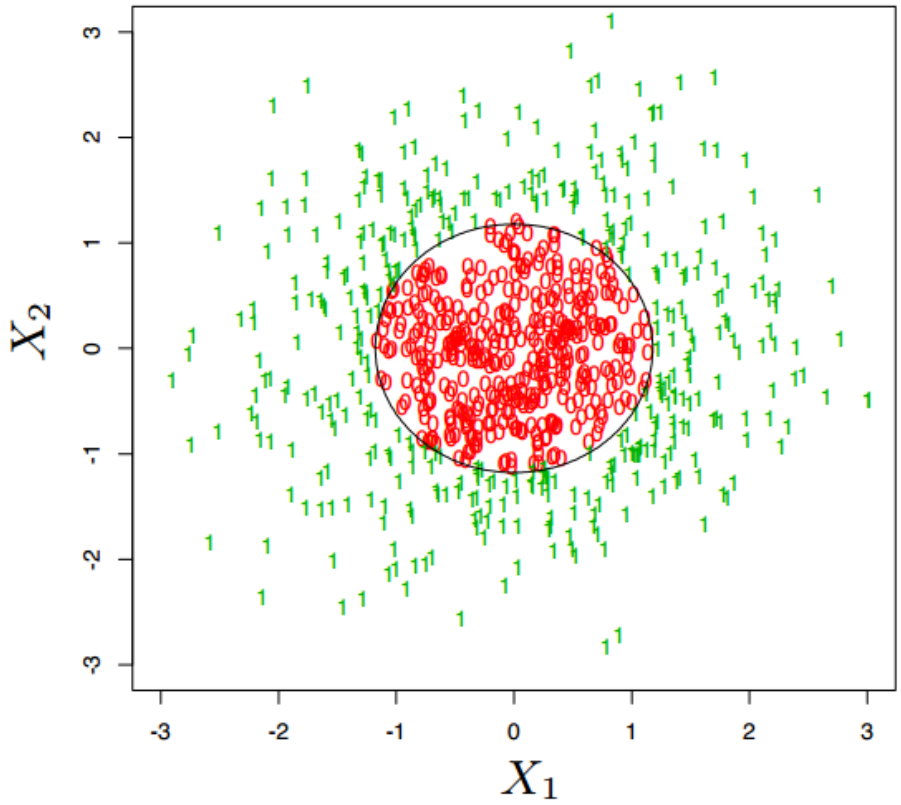


- SVM outperforms Decision Tree.
- AUC: Area Under Curve.

Function Class

- What functions can decision trees model?
  - Non-linear: very powerful function class
  - A decision tree can encode any Boolean function
  - Proof
    - Given a truth table for a function
    - Construct a path in the tree for each row of the table
    - Given a row as input, follow that path to the desired leaf (output)
- Problem: exponentially large trees!

Toy Example

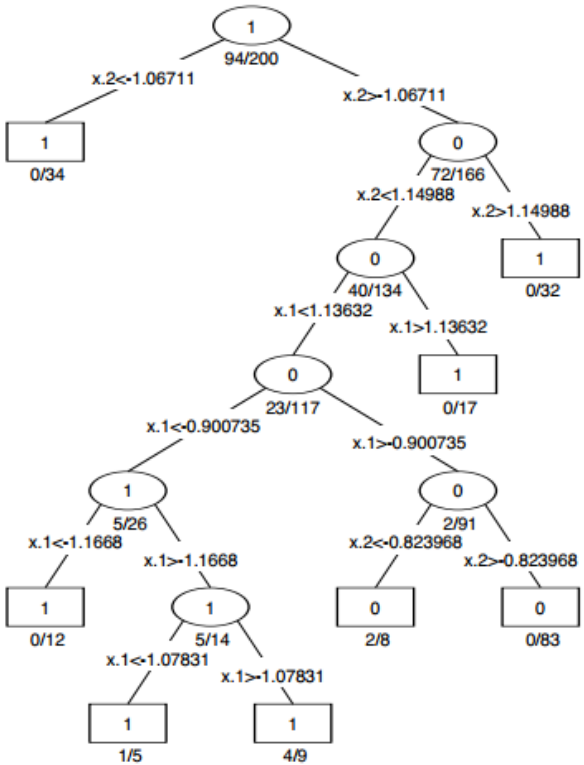


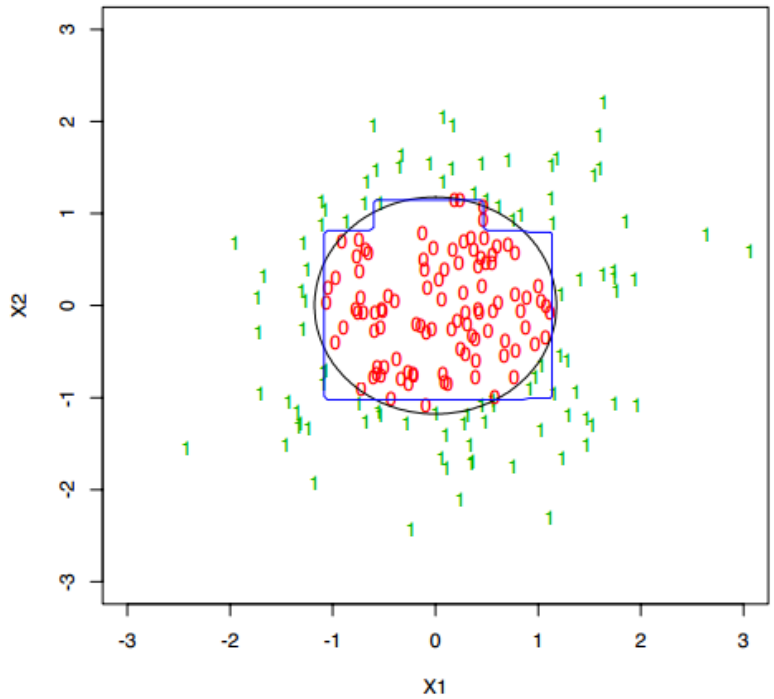
- Nonlinear separable data.
- Optimal decision boundary: $X_1^2 + X_2^2 = 1$.

Bagging and Random Forest

- **Bagging** (Bootstrap Aggregation) is used when our goal is to reduce the variance of a decision tree. Here idea is to create several subsets of data from training sample chosen randomly with replacement. Now, each collection of subset data is used to train their decision trees. As a result, we end up with an ensemble of different models. Average of all the predictions from different trees are used which is more robust than a single decision tree.
- **Random Forest** is an extension over bagging. It takes one extra step where in addition to taking the random subset of data, it also takes the random selection of features rather than using all features to grow trees. When you have many random trees. It's called Random Forest

Toy Example: Decision Tree





- Sample size: 200
- 7 branching nodes; 6 layers.
- Classification error: 7.3% when $d = 2$; $> 30\%$ when $d = 10$.

Model Averaging

Decision trees can be simple, but often produce noisy (bushy) or weak (stunted) classifiers.

- Bagging (Breiman, 1996): Fit many large trees to bootstrap-resampled versions of the training data, and classify by majority vote.
- Boosting (Freund & Shapire, 1996): Fit many large or small trees to reweighted versions of the training data. Classify by weighted majority vote.
- Random Forests (Breiman 1999): Fancier version of bagging.

In general Boosting>Random Forests>Bagging>Single Tree.

Bagging/Bootstrap Aggregation

Motivation: Average a given procedure over many samples to reduce the variance.

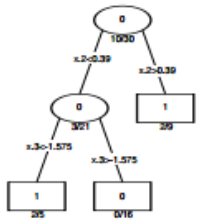
- Given a classifier $C(S, x)$, based on our training data S, producing a predicted class label at input point x.
- To bag C, we draw bootstrap samples $S_1, ..., S_B$ each of size N with replacement from the training data. Then

$$\widehat{C}_{\text{Bag}} = \text{Mojarity Vote}\{C(S_b, x)\}_{b=1}^{B}.$$

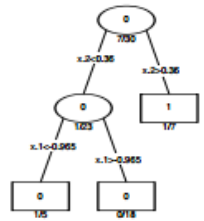
- Bagging can dramatically reduce the variance of unstable procedures (like trees), leading to improved prediction.
- All simple structures in a tree are lost.

Toy Example: Bagging Trees

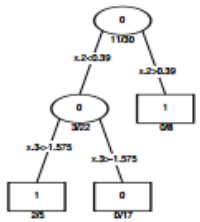
Original Tree



Bootstrap Tree 1

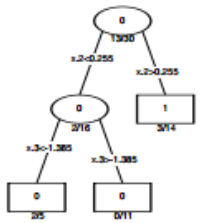


Bootstrap Tree 2

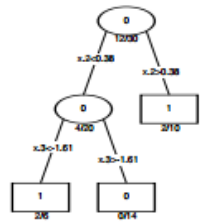


Bootstrap Tree 3

Bootstrap Tree 4

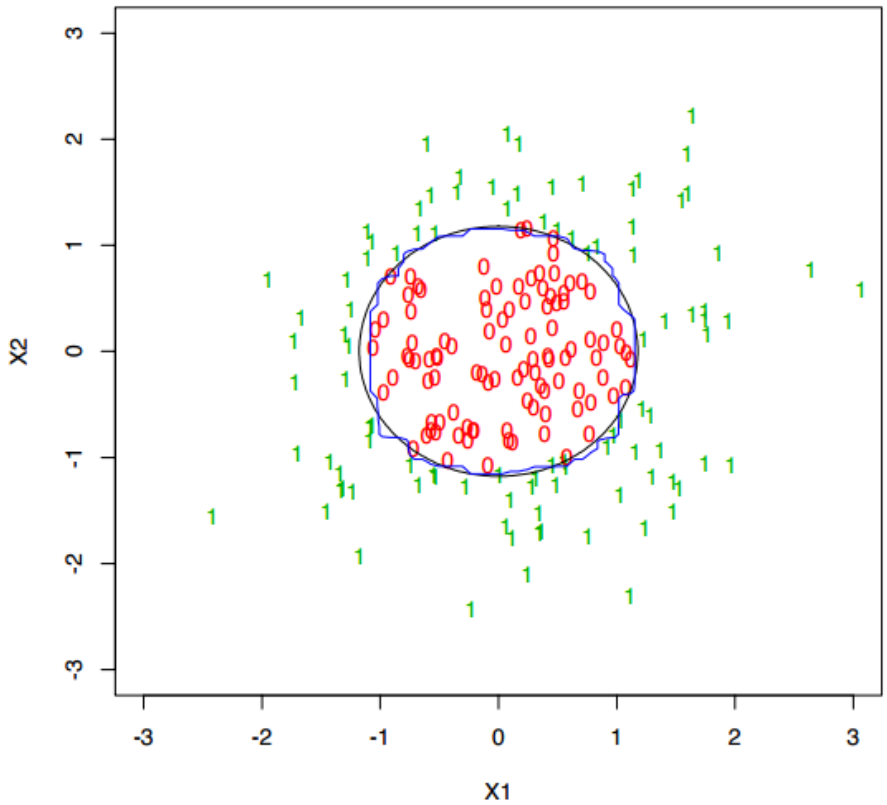


Bootstrap Tree 5



- 2 branching nodes; 2 layers.
- 5 dependent trees.

Toy Example: Bagging Trees



- A smoother decision boundary.
- Classification error: 3.2% (Single deeper tree 7.3%).

Random Forest

- Suppose there are N observations and M features in training data set. First, a sample from training data set is taken randomly with replacement.
- A subset of M features are selected randomly and whichever feature gives the best split is used to split the node iteratively.
- The tree is grown to the largest.
- Above steps are repeated and prediction is given based on the aggregation of predictions from n number of trees.

Random Forest

Advantages

- Handles higher dimensionality data very well.
- Handles missing values and maintains accuracy for missing data.

Disadvantages

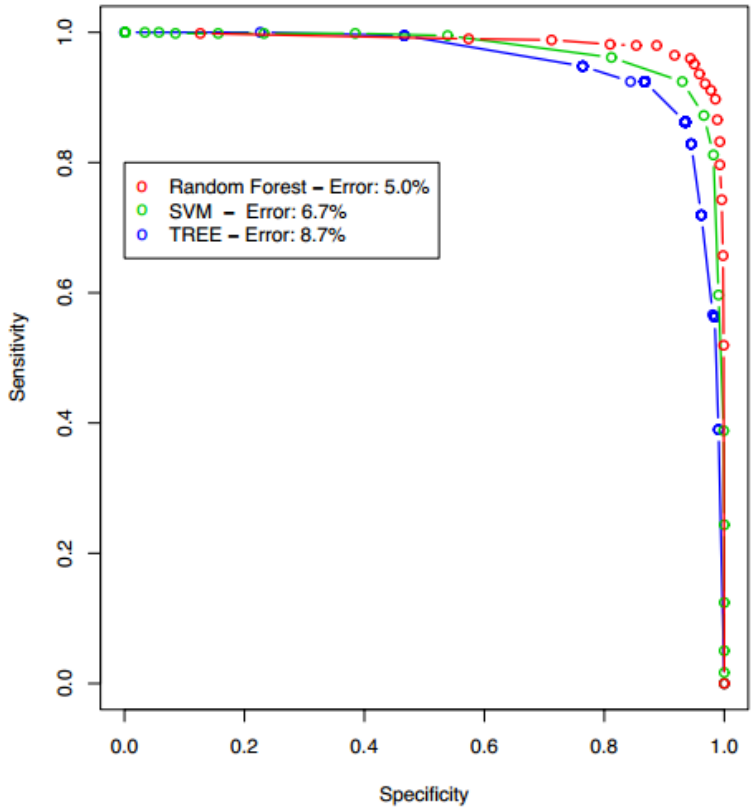
- Since final prediction is based on the mean predictions from subset trees, it won't give precise values for the regression model.

Random Forest

Bagging features and samples simultaneously:

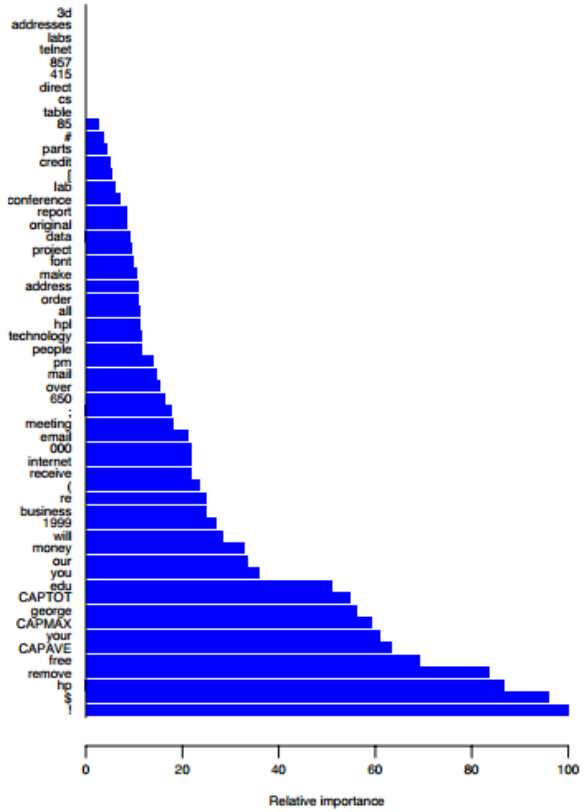
- At each tree split, a random sample of m features is drawn, and only those m features are considered for splitting. Typically $m = \sqrt{d}$ or $\log_2 d$, where d is the number of features
- For each tree grown on a bootstrap sample, the error rate for observations left out of the bootstrap sample is monitored. This is called the "out-of-bag" error rate.
- random forests tries to improve on bagging by "de-correlating" the trees. Each tree has the same expectation.

Random Forest for Spam Classification



- RF outperforms SVM.
- 500 Trees.

Random Forest: Variable Importance Scores



- The bootstrap roughly covers $1/3$ samples for each time.
- Do permutations over variables to check how important they are.

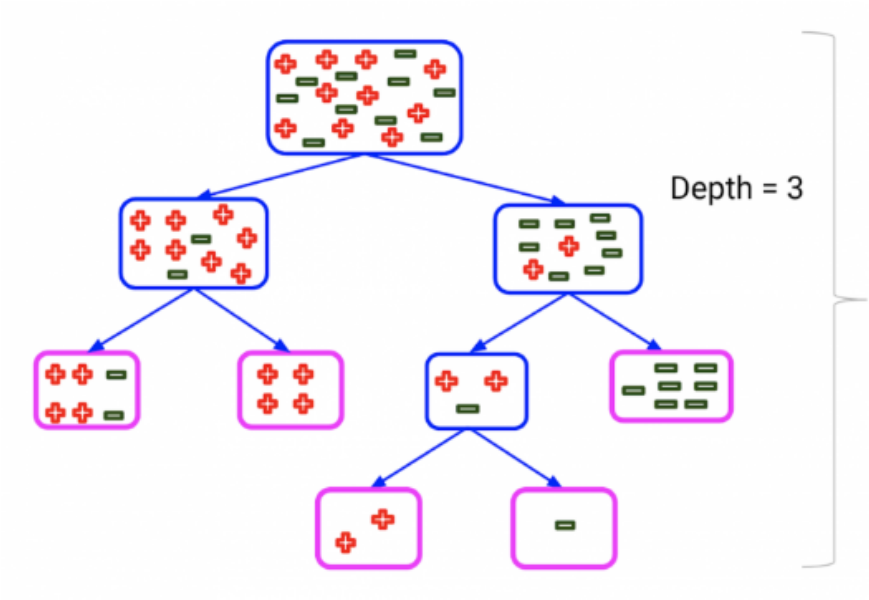
Random Forest Hyperparameters

- max_depth
- min_sample_split
- max_leaf_nodes
- min_samples_leaf
- n_estimators
- max_sample (bootstrap sample)
- max_features

Random Forest Hyperparameters

Random Forest Hyperparameter #1: max_depth

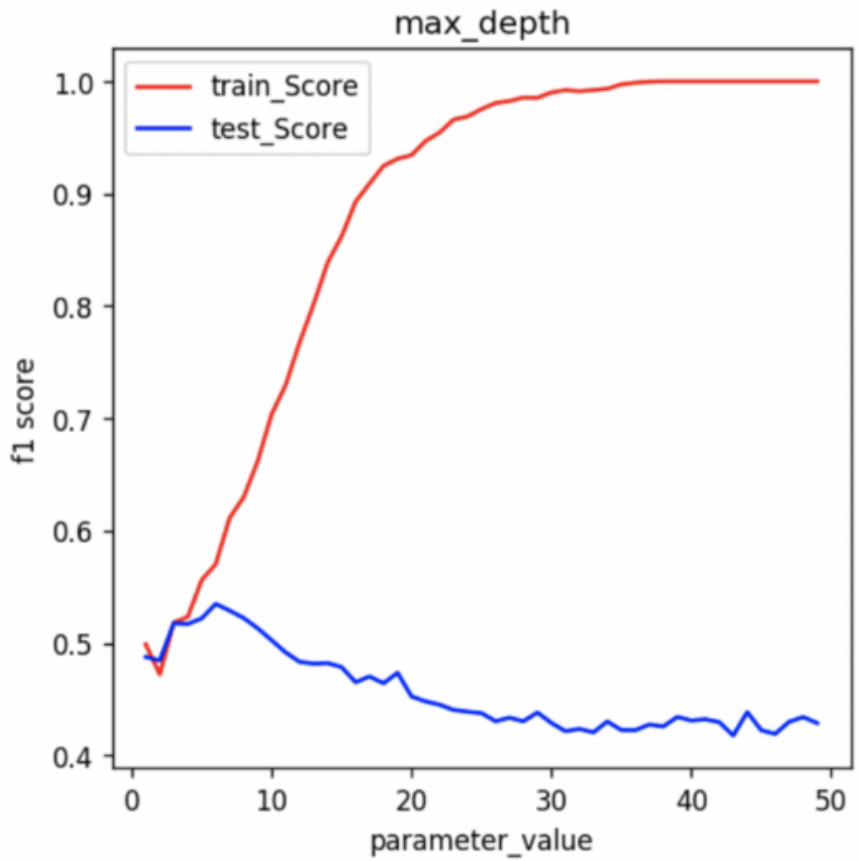
Let's discuss the critical *max_depth* hyperparameter first. The *max_depth* of a tree in Random Forest is defined as the longest path between the root node and the leaf node:



Using the *max_depth* parameter, I can limit up to what depth I want every tree in my random forest to grow.

https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

Random Forest Hyperparameters



In this graph, we can clearly see that as the max depth of the decision tree increases, the performance of the model over the training set increases continuously. On the other hand as the *max_depth* value increases, the performance over the test set increases initially but after a certain point, it starts to decrease rapidly.

Can you think of a reason for this? The tree starts to overfit the training set and therefore is not able to generalize over the unseen points in the test set.

Among the parameters of a decision tree, *max_depth* works on the macro level by greatly reducing the growth of the Decision Tree.

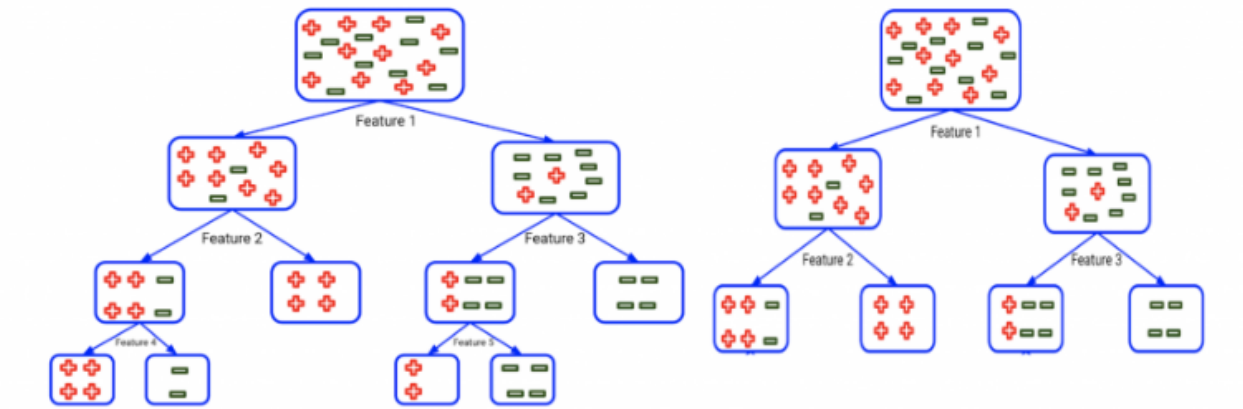
Random Forest Hyperparameters

Random Forest Hyperparameter #2: min_sample_split

> *min_sample_split – a parameter that tells the decision tree in a random forest the minimum required number of observations in any given node in order to split it.*

The default value of the minimum_sample_split is assigned to 2. This means that if any terminal node has more than two observations and is not a pure node, we can split it further into subnodes.

Having a default value as 2 poses the issue that a tree often keeps on splitting until the nodes are completely pure. As a result, the tree grows in size and therefore overfits the data.

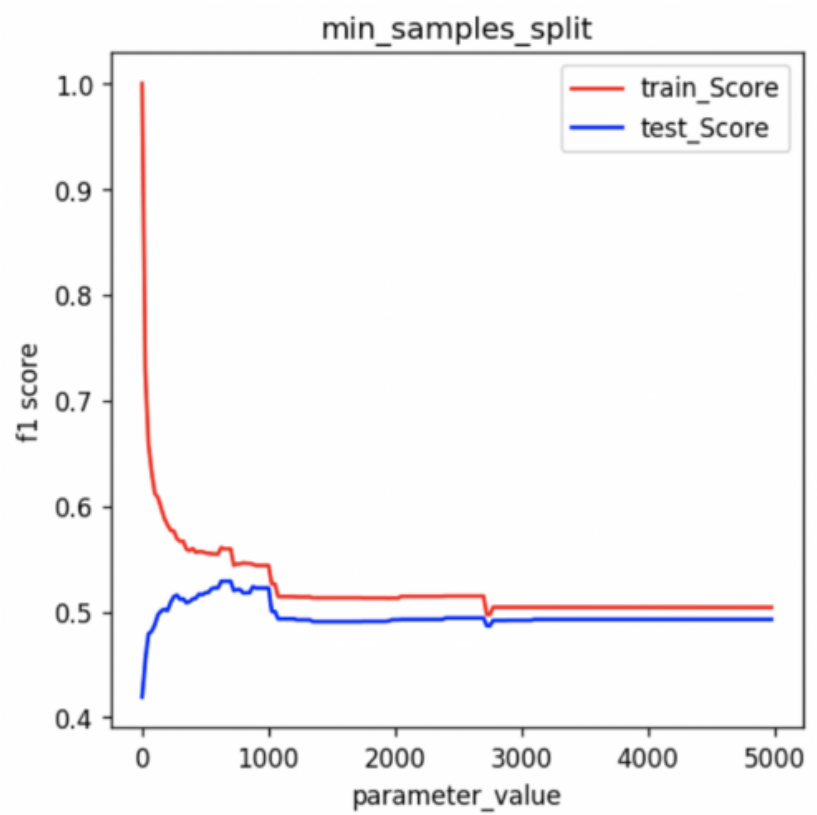


By increasing the value of the *min_sample_split*, we can reduce the number of splits that happen in the decision tree and therefore prevent the model from overfitting. In the above example, if we increase the *min_sample_split* value from 2 to 6, the tree on the left would then look like the tree on the right.

https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

Random Forest Hyperparameters

Now, let's look at the effect of *min_samples_split* on the performance of the model. The graph below is plotted considering that all the other parameters remain the same and only the value of *min_samples_split* is changed:



On increasing the value of the *min_sample_split* hyperparameter, we can clearly see that for the small value of parameters, there is a significant difference between the training score and the test scores. But as the value of the parameter increases, the difference between the train score and the test score decreases.

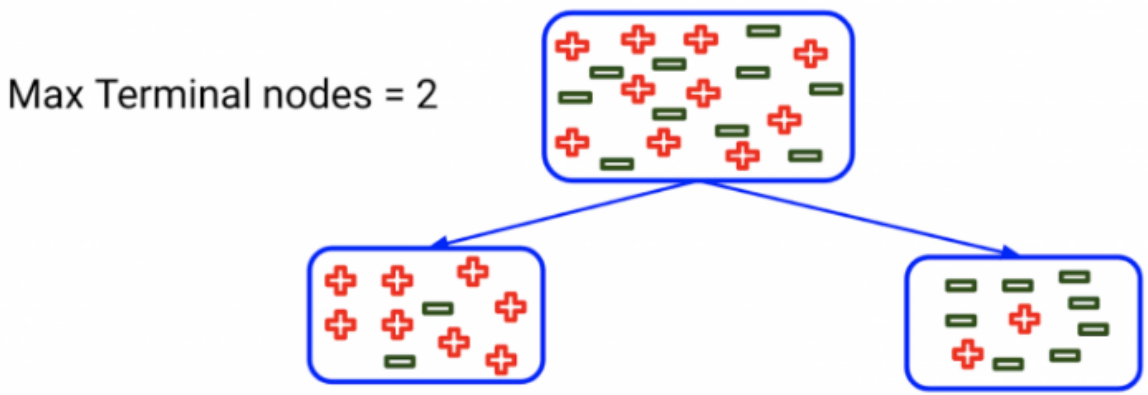
But there's one thing you should keep in mind. *When the parameter value increases too much, there is an overall dip in both the training score and test scores. This is due to the fact that the minimum requirement of splitting a node is so high that there are no significant splits observed. As a result, the random forest starts to underfit.*

Random Forest Hyperparameters

Random Forest Hyperparameter #3: max_terminal_nodes

Next, let's move on to another Random Forest hyperparameter called *max_leaf_nodes*. **This hyperparameter sets a condition on the splitting of the nodes in the tree and hence restricts the growth of the tree.** If after splitting we have more terminal nodes than the specified number of terminal nodes, it will stop the splitting and the tree will not grow further.

Let's say we set the maximum terminal nodes as 2 in this case. As there is only one node, it will allow the tree to grow further:

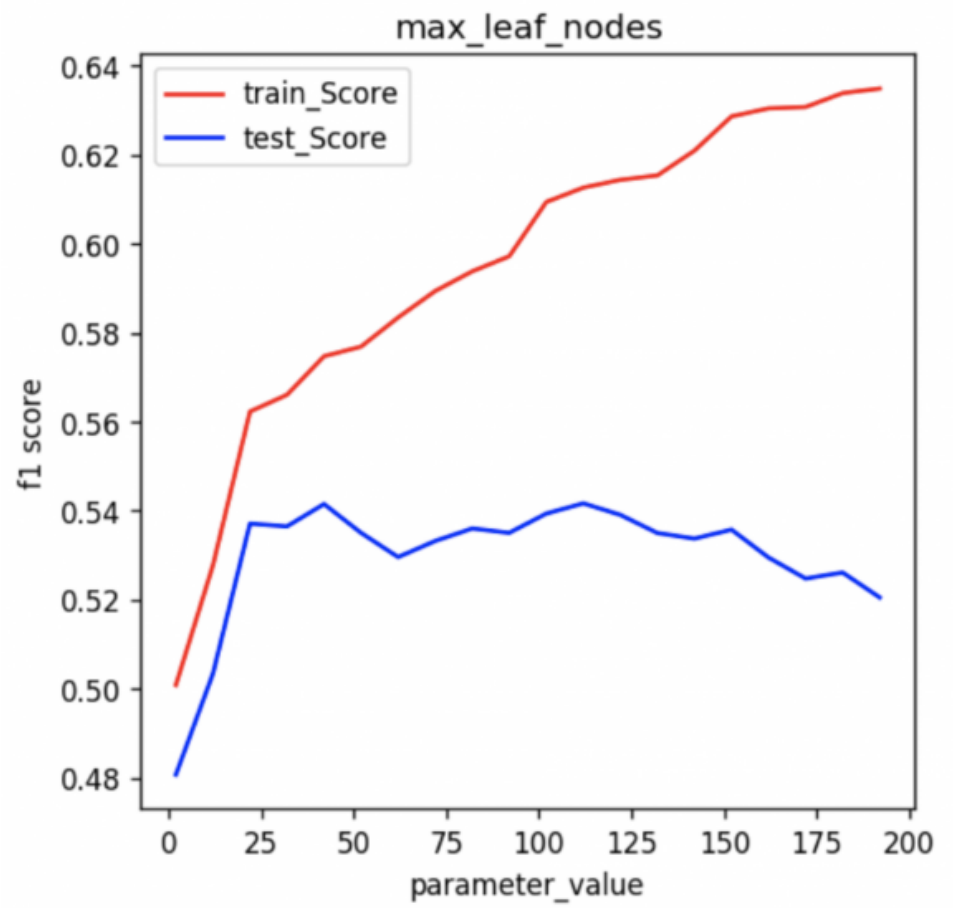


Now, after the first split, you can see that there are 2 nodes here and we have set the maximum terminal nodes as 2. Hence, the tree will terminate here and will not grow further. This is how setting the maximum terminal nodes or *max_leaf_nodes* can help us in preventing overfitting.

https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

Random Forest Hyperparameters

Note that if the value of the *max_leaf_nodes* is very small, the random forest is likely to underfit. Let's see how this parameter affects the random forest model's performance:



We can see that when the parameter value is very small, the tree is underfitting and as the parameter value increases, the performance of the tree over both test and train increases. According to this plot, the tree starts to overfit as the parameter value goes beyond 25.

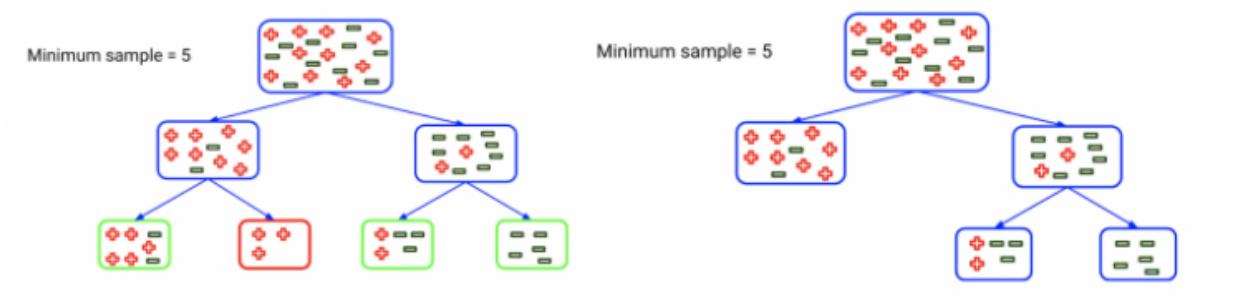
https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

Random Forest Hyperparameters

Random Forest Hyperparameter #4: min_samples_leaf

Time to shift our focus to *min_sample_leaf*. This Random Forest hyperparameter specifies the minimum number of samples that should be present in the leaf node **after splitting** a node.

Let's understand *min_sample_leaf* using an example. Let's say we have set the minimum samples for a terminal node as 5:



The tree on the left represents an unconstrained tree. Here, the nodes marked with green color satisfy the condition as they have a minimum of 5 samples. Hence, they will be treated as the leaf or terminal nodes.

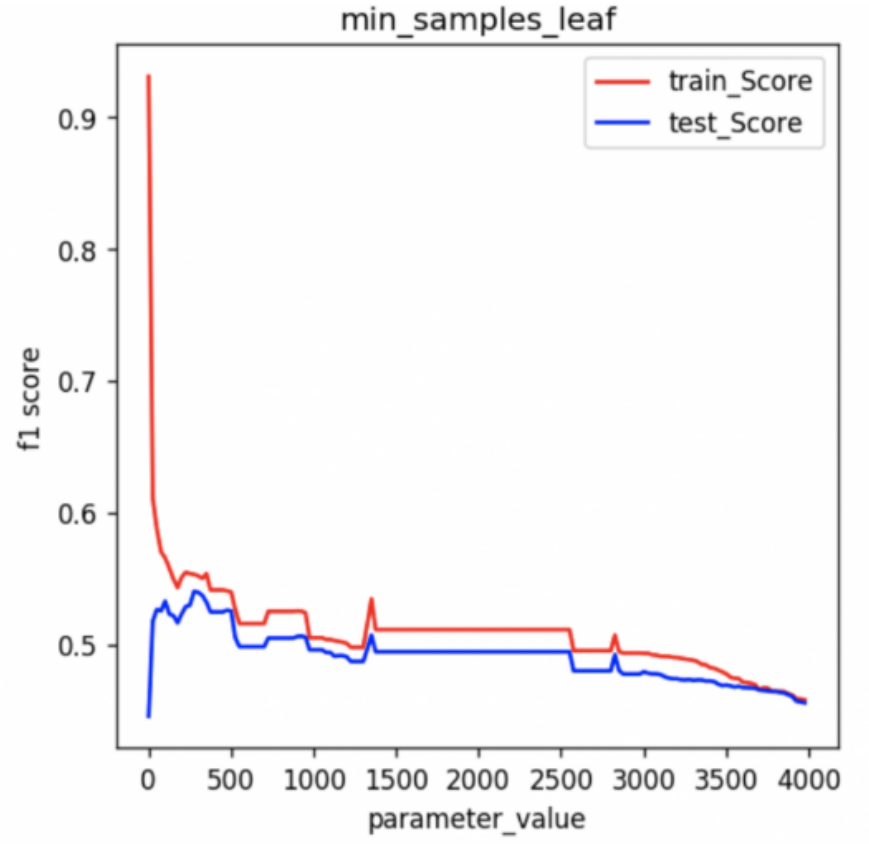
However, the red node has only 3 samples and hence it will not be considered as the leaf node. Its parent node will become the leaf node. That's why the tree on the right represents the results when we set the minimum samples for the terminal node as 5.

So, we have controlled the growth of the tree by setting a minimum sample criterion for terminal nodes. As you would have guessed, similar to the two hyperparameters mentioned above, this hyperparameter also helps prevent overfitting as the parameter value increases.

https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

Random Forest Hyperparameters

If we plot the performance/parameter value plot as before:



We can clearly see that the Random Forest model is overfitting when the parameter value is very low (when parameter value < 100), but the model performance quickly rises up and rectifies the issue of overfitting (100 < parameter value < 400). But when we keep on increasing the value of the parameter (> 500), the model slowly drifts towards the realm of underfitting.

https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

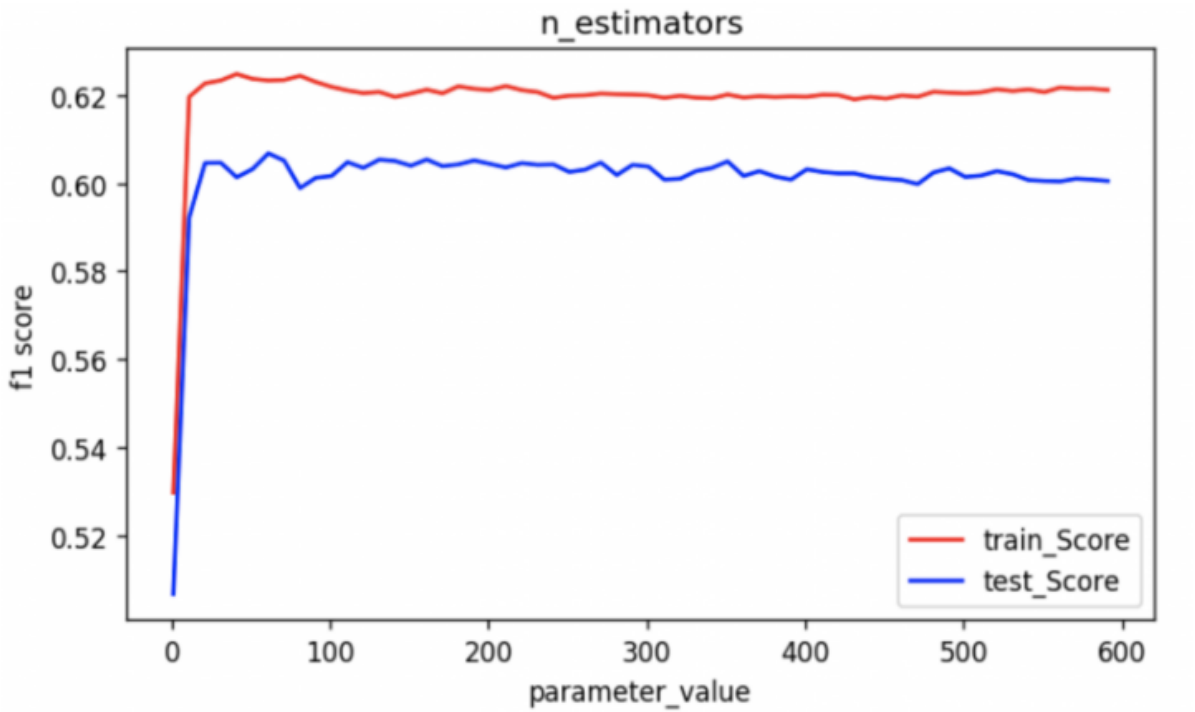
Random Forest Hyperparameters

Random Forest Hyperparameter #5: n_estimators

We know that a Random Forest algorithm is nothing but a grouping of trees. But how many trees should we consider? That's a common question fresher data scientists ask. And it's a valid one!

We might say that more trees should be able to produce a more generalized result, right? But by choosing more number of trees, the time complexity of the Random Forest model also increases.

In this graph, we can clearly see that the performance of the model sharply increases and then stagnates at a certain level:

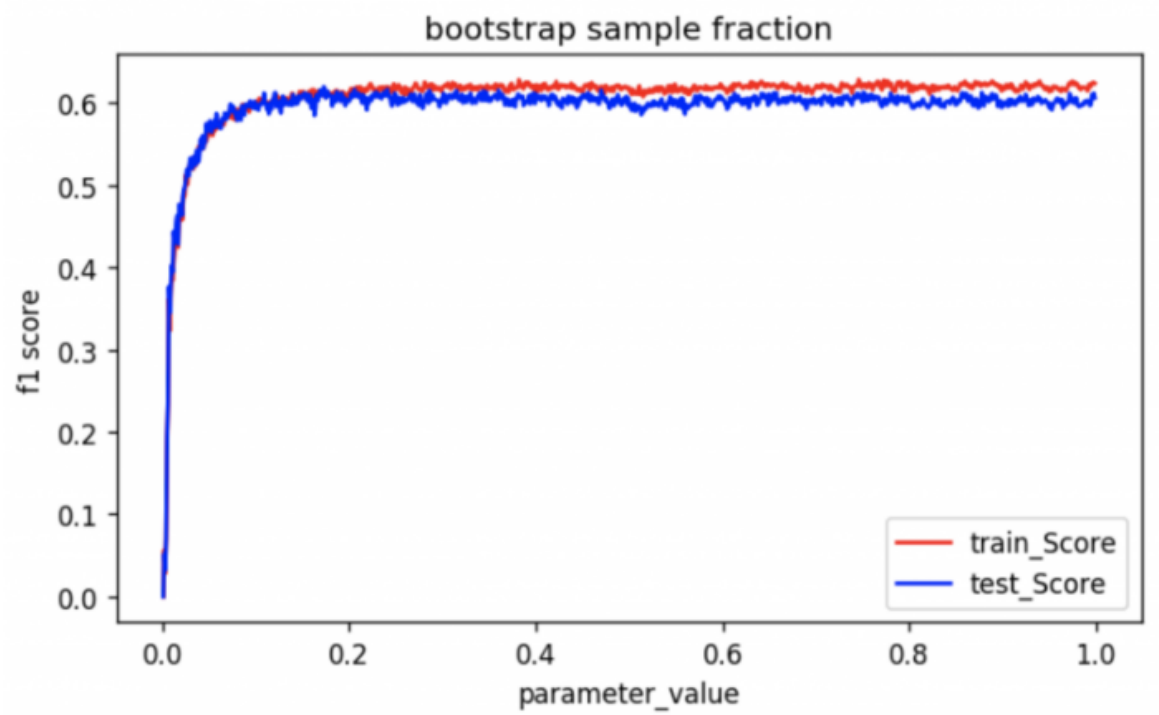


This means that choosing a large number of estimators in a random forest model is not the best idea. Although it will not degrade the model, it can save you the computational complexity and prevent the use of a fire extinguisher on your CPU!

Random Forest Hyperparameters

Random Forest Hyperparameter #6: max_samples

The *max_samples* hyperparameter determines what fraction of the original dataset is given to any individual tree. You might be thinking that more data is always better. Let's try to see if that makes sense.



We can see that the performance of the model rises sharply and then saturates fairly quickly. Can you figure out what the key takeaway from this visualization is?

It is not necessary to give each decision tree of the Random Forest the full data. If you would notice, the model performance reaches its max when the data provided is less than 0.2 fraction of the original dataset. That's quite astonishing!

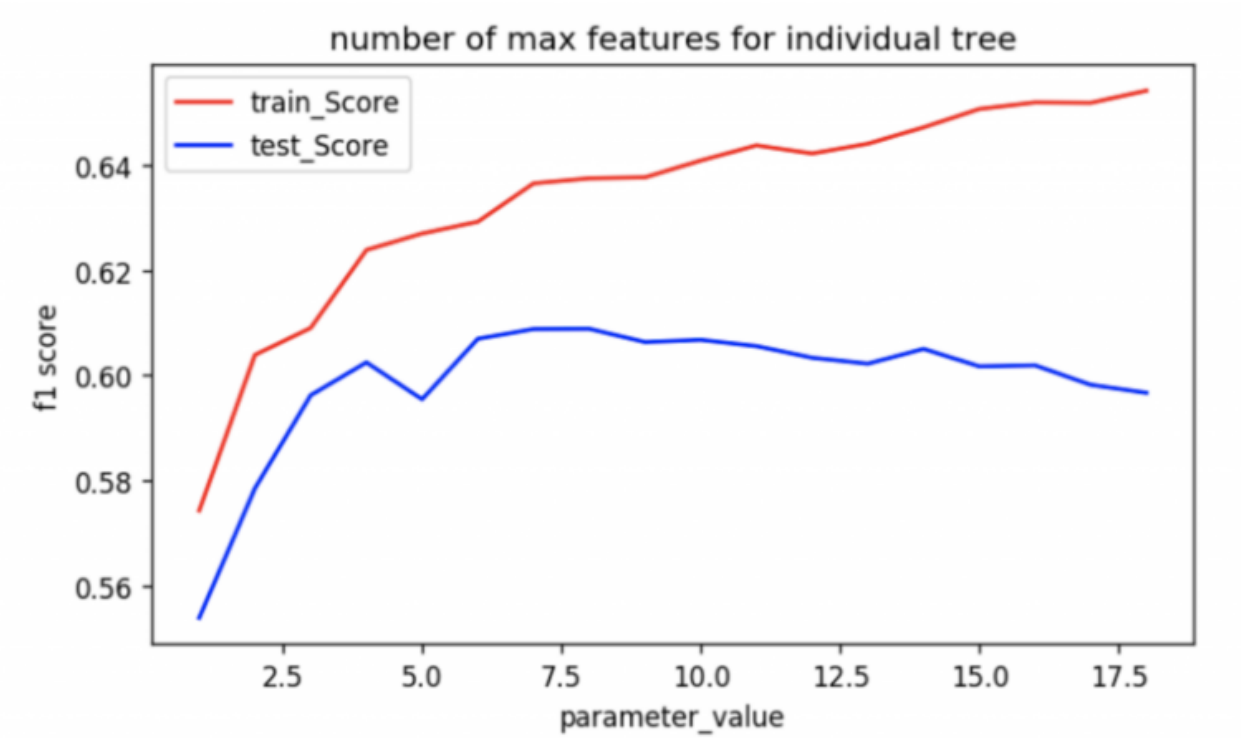
Although this fraction will differ from dataset to dataset, we can allocate a lesser fraction of bootstrapped data

Random Forest Hyperparameters

Random Forest Hyperparameter #7: max_features

Finally, we will observe the effect of the *max_features* hyperparameter. This resembles the number of maximum features provided to each tree in a random forest.

We know that random forest chooses some random samples from the features to find the best split. Let's see how varying this parameter can affect our random forest model's performance.



We can see that the performance of the model initially increases as the number of *max_feature* increases. But, after a certain point, the *train_score* keeps on increasing. But the *test_score* saturates and even starts decreasing towards the end, which clearly means that the model starts to overfit.

Ideally, the overall performance of the model is the highest close to 6 value of the max features. It is a good

Gradient Boosting

- Gradient Boosting = Gradient Descent + Boosting
- It uses gradient descent algorithm which can optimize any differentiable loss function. An ensemble of trees are built one by one and individual trees are summed sequentially. Next tree tries to recover the loss (difference between actual and predicted values).

Decision Tree or Random Forest

- Suppose a bank has to approve a small loan amount for a customer and the bank needs to make a decision quickly. The bank checks the person's credit history and their financial condition and finds that they haven't re-paid the older loan yet. Hence, the bank rejects the application.
- But here's the catch – the loan amount was very small for the bank's immense coffers and they could have easily approved it in a very low-risk move. Therefore, the bank lost the chance of making some money.
- Now, another loan application comes in a few days down the line but this time the bank comes up with a different strategy – multiple decision-making processes. Sometimes it checks for credit history first, and sometimes it checks for customer's financial condition and loan amount first. Then, the bank combines results from these multiple decision-making processes and decides to give the loan to the customer.

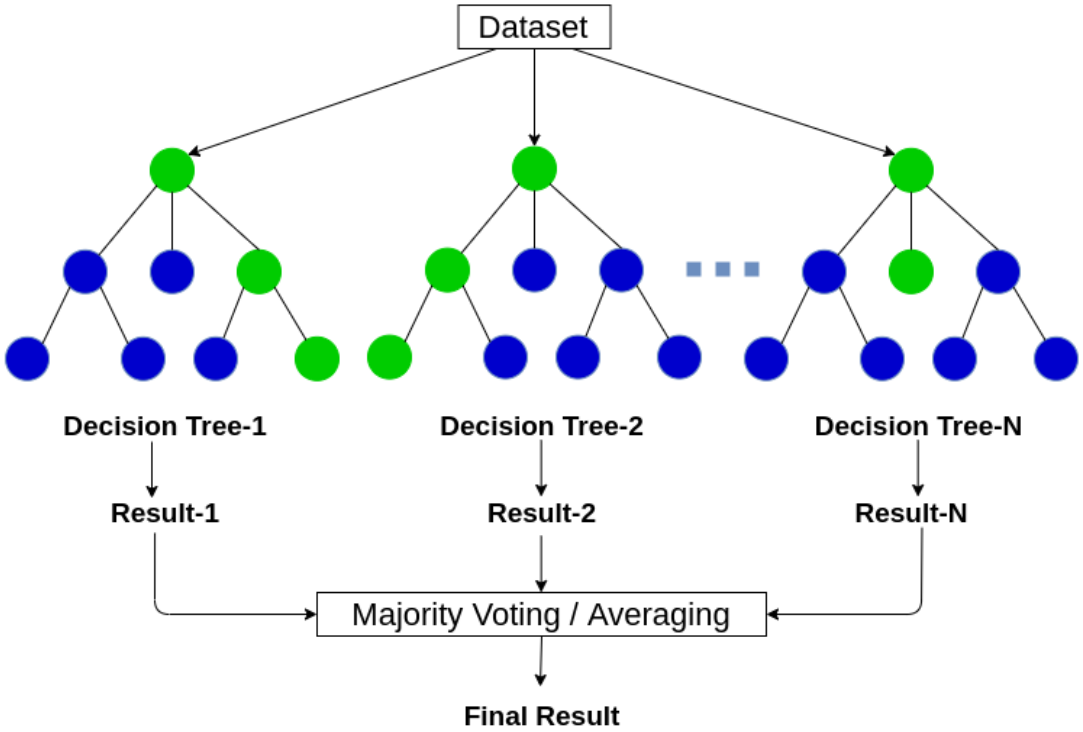
https://www.analyticsvidhya.com/blog/2020/05/

Decision Tree or Random Forest

- Even if this process took more time than the previous one, the bank profited using this method. This is a classic example where collective decision making outperformed a single decision-making process.
- Now, here's my question to you – do you know what these two processes represent?

`https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/`

Random Forest



https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/