CS337 , Fall 2023
Regularized Logistic Regression, Decision Tree Classification

Scribes: Isha Arora*, Harshit Agarwal*, Pampa Sow Mondal*
Bijili Prachothan Varma*, Gorle Krishna Chaitanya*, Madur Adarsh Reddy*
Aditya Singh, Parth Pujari, Nenavath Preetham
Yashwanth Reddy Challa, Aditya Yadav, Aditya Rao Gulbarga
Edited by: Barah Fazili

August 29 and August 31, 2023

1 Logistic Regression with Regularization

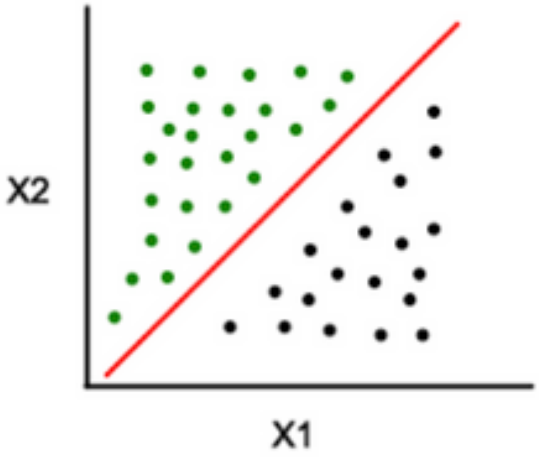


Figure 1: Linearly separable data

*Most of the content was derived from their submission.

Logistic Regression is capable of perfectly classifying linearly separable data. But there may be mulitple weight vectors that can separate the data. Let us consider the case in which the dataset is linearly separable and the true decision boundary is represented by the straight line:

$$x_1 - 2x_2 = 0$$

as shown in the above figure.
The logistic regression model while predicting the decision boundary, may predict it as

$$x_1 - 2x_2 = 0$$

and thus

$$w_1 = 1,\ w_2 = -2$$

or the values of w_1 and w_2 might be arbitrarily scaled up by the same amount (the decision boundary remains unchanged with arbitrary equal scaling) For instance, the weights predicted by the logistic regression model might be

$$w_1 = 10,\ w_2 = -20$$

or

$$w_1 = 10^5,\ w_2 = -2 * 10^5$$

Given this one might wonder what are the values of w that the logistic regression classifier is likely to converge at?
We know that:

$$P(y_i = 1|x_i, w) = \sigma(w^T x) = \frac{1}{1 + exp(-w^T x)}$$

The aim of the classifier is to maximize the above probability for the points in the training dataset and thus it is likely that it will scale up the values of the weights to decrease the value of

$$exp(-w^T x)$$

and thus increasing the value of

$$P(y_i = 1|x_i, w)$$

to about 1 for all of the positive samples. Hence it is likely that this (unregularized) classifier would choose high values of the weights w.
But is this ideal?

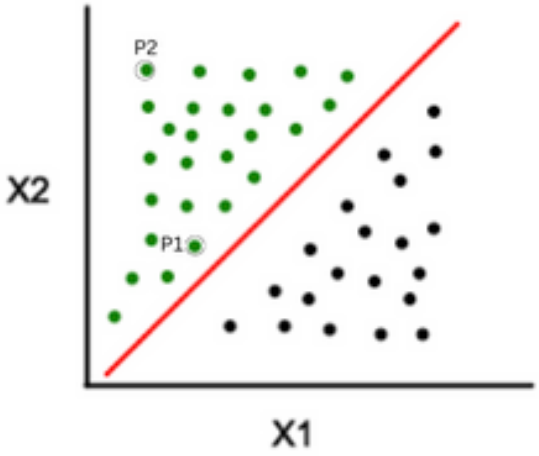


Figure 2: See points P1 and P2

Consider two positive sample points as shown in the above figure, one very close to the decision boundary, P1 and the other further away from the decision boundary, P2. We would ideally want a higher value of

$$P(y_i = 1|x_i, w) = \sigma(w^T x) = \frac{1}{1 + exp(-w^T x)}$$

for P2 than P1 as it is far away from the decision boundary and so prediction confidence should be higher for this point than P1. But when the values of the coefficients w in logistic regression are arbitrarily high, it diminishes the distinction between data points located near the decision boundary and those far from it as the probability for all the positive samples nearly tends to 1. This occurs because the exponential term in the logistic function causes the probabilities assigned to both types of points to become substantially higher.

To address this issue, we introduce a **regularization term** along with the loss function.

$$w_R^* = \arg\min_w \; L_{CE}(w, D) + \lambda ||w||_2^2$$

The regularization serves to constrain the values of w, preventing them from becoming arbitrarily high. By applying regularization, we effectively limit the coefficients' magnitudes, which helps maintain the proper distinction between data points by preventing the model from assigning overly confident probabilities to points near the decision boundary.

The regularization penalty term forces the model to find a balance between minimizing the cross entropy loss (fitting the data) and minimizing the regularization term (keeping coefficients small). Regularization also ensures the model predictions are less sensitive to minor fluctuations or noise in the training data. Consequently, the above model becomes more robust and less likely to overfit, leading to a **reduction in variance**.

In logistic regression, when the decision boundary is linear, it is relatively straightforward to interpret the model. The feature interpretability of logistic regression models with linear decision boundaries is high

which means we can easily assess the importance and relevance of each feature or attribute in making predictions. However, when dealing with non-linear decision boundaries, especially in models with numerous features, interpreting the model and selecting the most informative features becomes much more challenging.

There are two desired features of classification models:

1. The ability to learn complex decision boundaries.
2. Interpretability of the model i.e. understand how useful are each of the features/attributes.

Non-linear logistic regression models aren't easily interpretable and next, we study another kind of classifiers which are Decision Tree Classifiers.

2 Learning Complex decision boundary: Decision Trees

Decision tree is an interpretable model whose final prediction can be written as a **disjunction of conjunctions** based on the attribute values over training instances.

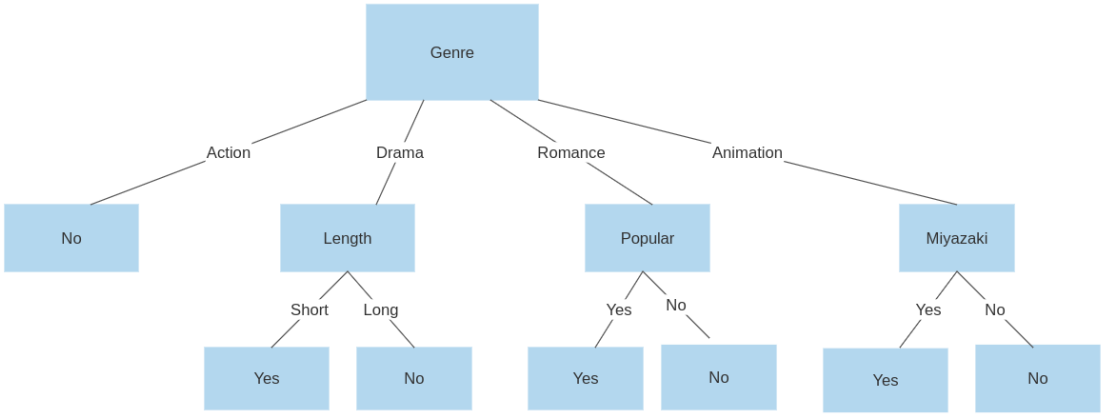


Figure 3: A decision tree to classify movies as liked (represented by 'Yes') or disliked (represented by 'No'). Each node of the tree represents an attribute and each path of the tree represents a conjunction of attributes and the final decision is a disjunction of these conjunctions. For instance for the above tree, the final prediction of the model can be written as $(Genre = Drama \wedge Length = Short) \vee (Genre = Romance \wedge Popular = Yes) \vee (Genre = Animation \wedge Mizayaki = Yes)$. The expression evaluates to True (yields the value 1) for a 'Yes' instance and evaluates to False (yields the value 0) for a 'No' instance.

2.1 Decision Boundaries of Decision Trees

Consider the following xor-like classified dataset in the above figure. The following might be a decision tree for the classification of this dataset.

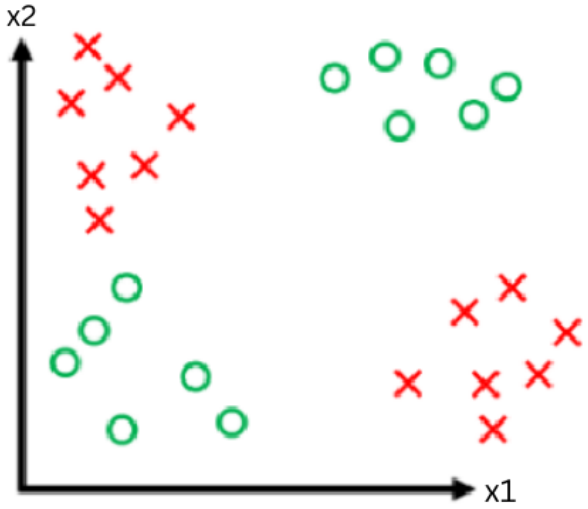


Figure 4: The x and y axis correspond to 2 attributes of the dataset, $x1$ and $x2$ respectively. The datapoints marked by green circles and the datapoints marked by red crosses represent 2 different categories (y_{labels}) say category 1 and category 2 respectively.

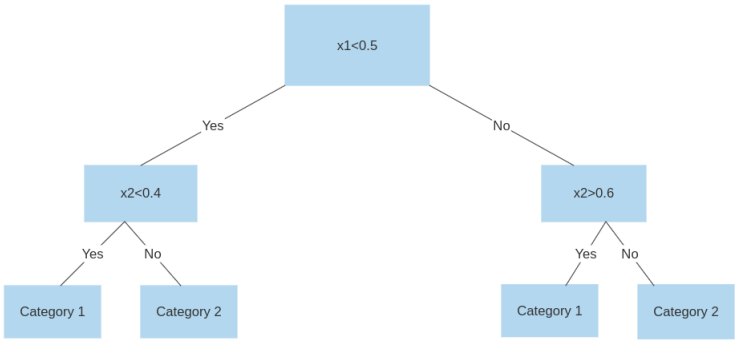


Figure 5: If the condition in the topmost decision node of the decision tree which is '$x_1 < 0.5$' evaluates to true, we move along the left edge of the tree (right otherwise). If the condition in the node at this level further evaluates to True, we move along the left edge again and predict category 1(represented by green circles).

Thus, on the basis of the above decision tree, the decision boundaries of the decision tree are shown in the figure below.
Thus, as seen above if nodes of the decision tree depend on a single attribute only, then decision trees divide the feature space into **hyper - rectangles** or equivalently we can say that the resulting decision boundaries are axis-parallel hyper-planes.

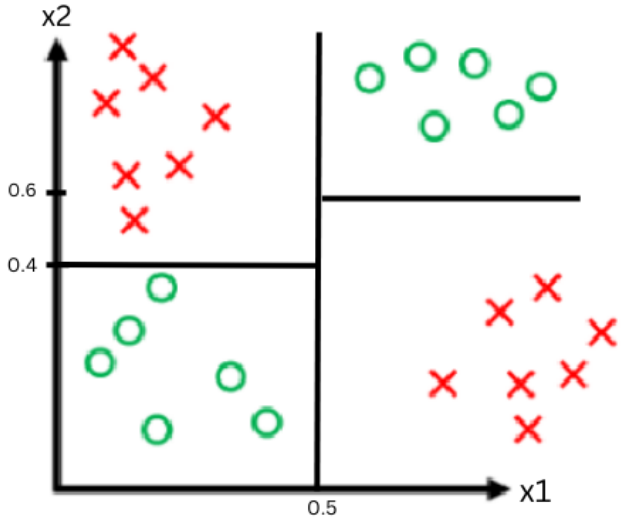


Figure 6: The vertical line represents the topmost decision node of the decision tree that is '$x_1 < 0.5$' and the two horizontal lines represent $x_2 < 0.4$' and '$x_2 > 0.6$' respectively. We can see that the decision boundaries are just the depiction of the nodes (the attributes based on which decisions were taken) of the decision tree when plotted on a graph.

NOTE: Instead of axis-parallel hyperplanes, we can even have linear hyper-planes. In the later case, instead of the decision nodes being a linear function of a single attribute, they can be a linear function of 2 or more attributes, say for instance $x_1 + x_2 > 0$. Thus, in such cases instead of axis-parrallel decision boundaries, we would have locally-linear decision boundaries. Typically, axis-parallel hyperplanes are used for decision tree modelling.

2.2 Finding the optimal Decision Tree

Unlike the case for logistic regression, finding the smallest Decision Tree that is optimal with respect to some metric (like cross entropy loss etc.) involving all of the attributes is an **NP-hard problem**.
Decision tree estimation is mostly done **greedily** in which the tree is built recursively.

SIMPLE DECISION TREE TEMPLATE:

- Start from an empty node with all instances.
- Pick the **best** attribute to split on. For instance in figure 1, genre was chosen as the first attribute
- Repeat step 2 recursively for each new node until a **stopping criterion** is met

Now, two important questions arise:

- What is the notion of best attribute and how do we find it?
- What is a good stopping criterion?

It is to be noted that a lot of heuristics are involved in decision tree construction in contrast to logistic regression where we have a clearly formulated optimisation problem and a final solution to the optimisation problem.
We will start by reflecting on how to choose the best attribute.

2.3 Choosing node attributes

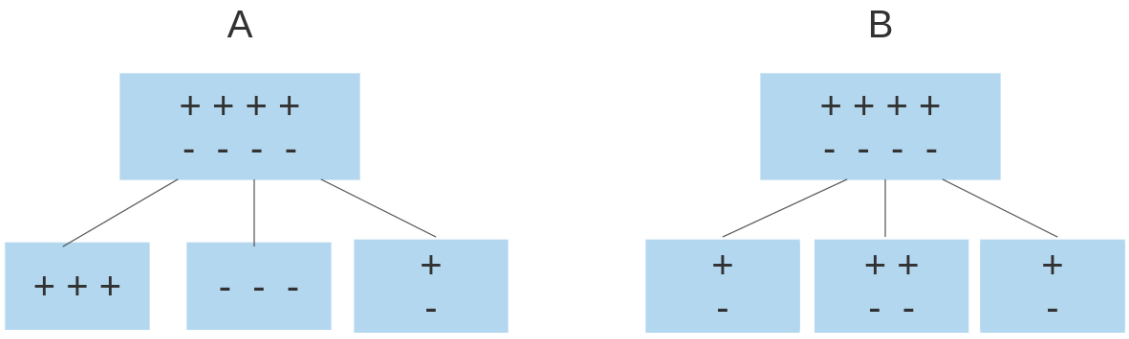


Figure 7: Unlike the previous trees we have seen, the nodes of the above tree represent the datapoints that would reach these nodes based on the values of some attribute. In both of the above images, the topmost node represents the entire dataset (as no division of the dataset on the basis of some attribute has been done yet). Then on the basis of some attribute which is clearly different for A and B, the dataset is divided into 3 datasets which are present in the three child nodes of each parent.

Seeing the above classification, it seems intuitive that the attribute on which decision has been taken in A is better than the one in B. This is because, A's attribute is leading to nodes that are already largely **homogeneous** and thus the attribute in A immediately effectively reduces the length of the decision tree. On extending the tree, B might later on have better accuracy but the top attribute is just overly building out the tree which is not preferable as overly large trees might lead to overfitting. Thus, our goal somewhat broadly is find simple models that generate good subtrees that get added on and we want the size of subtrees to be as small as possible.
Intuition for a *good* split (attribute): A good split for an attribute results in subsets that are (mostly) entirely homogeneous that is the dataset in each split is (mostly) all of the same category.
Now, we need a quantitative way to suit our intuitions for a good attribute and thus we introduce the following concepts.

2.4 Entropy

Entropy of a random variable is a measure of **uncertainty** in the value of that random variable. Let X be a random variable and x represent a particular value of the random variable. Entropy of X is H(X) where H(X) is defined as

$$H(X) = -\sum_{x} P(X = x) \log_2(P(X = x)$$

To understand what $H(X)$ signifies, let's pick up our familiar coin toss example in which the random variable can take the value 1 or 0 on the basis of the result of the coin toss (heads or tails respectively). $H(X)$ for this random variable is plotted below:

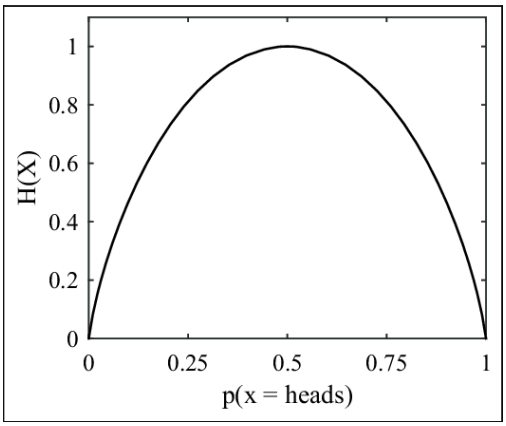


Figure 8: Entropy of coin toss

If the coin is fair (P(Head) = P(Tail) = 0.5) then there is maximum uncertainty in toss outcome and thus entropy has a maxima at this point and if the coin is heavily biased (e.g P(Head)= 0.9, P(Tail)= 0.1) then there is some certainty about whether it would be head or tail and thus entropy is low. Thus, higher the uncertainty in the outcome of the random variable, higher is the entropy. Thus it can be said that,

- A high value of entropy asserts a nearly uniform distribution . We do not know the next outcome(e.g coin toss).
- A lower value of Entropy asserts that the distribution of the random variable has well-defined modes and thus there is some certainty in the outcome of the random variable.

ENTROPY OF A DATASET Entropy of a dataset measures the uncertainty in the group of observations. Consider a dataset S with k classes/labels. Entropy of the dataset S is $H(S)$ where $H(S)$ is defined as

$$H(S) = -\sum_{i=1}^{k} P_{i,S} \log(P_{i,S})$$

where $P_{i,S}$ is the relative count of instances in S with label i (the probability of randomly selecting an example of class i in S).
What happens when all instances belong to the same class? $p_{i,s}$ is 1 which implies that entropy of the dataset is 0.

2.5 Splitting Criterion: Information Gain

We are building towards deciding on how to choose the best attributes to build our decision tree such that when the data is split on their basis, we achieve maximum possible homogeneity in other words the maximum

drop in the entropy within two tree levels.
Information gain, is simply the reduction in entropy caused by partitioning the dataset S according to a particular attribute. It determines the quality of splitting and helps to determine the order of attributes in the nodes of a decision tree. Gain of a dataset S for a attribute $'a'$ is *Gain(s,a)* where $Gain(S, a)$ is defined as

$$Gain(S, a) = H(S) - \sum_{v \in V(a)} \frac{|S_v|}{|S|} H(s_v)$$

where $H(S)$ is the entropy of the dataset before spliting on the basis of the attribute $'a'$, S_v is the subset of dataset S whose instances all have the attribute $'a'$ taking the value $'v'$, and thus the term subtracted from $H(S)$ is nothing but the weighted sum of the entropies of the dataset into which the dataset S is split on the basis of attribute $'a'$. Thus for each node of the decision tree, we find the attribute with the maximum information gain and split on its basis in our decision tree.
Recap: A decision tree (DT) is a supervised learning method mainly used for classification tasks. It has a tree structure, with input for classification given at the root and the classification label of the input is decided based on the leaf node it ends up at. An input reaching a node is sent to one of the child nodes based on a certain condition on the input. A label of a node is defined as the maximum occurring label of the instances that reach the node (in training set).
Lets take an example

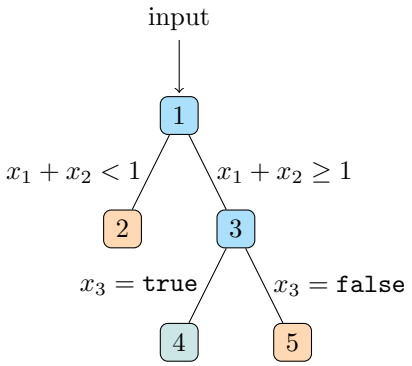


Figure 9: A simple DT, classifying the input based on 3 attributes. Nodes 2,4 and 5 are the leaf nodes.

Note Although node 1 of the DT in the figure above uses a combination of 2 attributes (x_1, x_2) in the condition, in practice, we usually split using condition on only one attribute. This is because it becomes very complex to consider all possible combinations (why only $x_1 + x_2$?, why not $x_1 x_2$, $\log x_1 + x_2$, etc...) of attributes while training. We can take combinations of input attributes if we know some priors on the dataset.

To train the DT we have to answer two main questions:

1. What is the best attribute to split the data?
2. When should we stop building the decision tree?

3 Information Gain

Finding the optimal way of splitting the data is NP-hard. So we resort to using heuristic greedy strategies to get a reasonably good split.

Heuristic is a technique designed for problem solving more quickly when classic methods are too slow for finding an exact or approximate solution, or when classic methods fail to find any exact solution. This is achieved by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut.

One common heuristic measure used in the context of decision trees is the **information gain**. Before we go to information gain, we have to define **entropy**. Consider a dataset S. Each element $s \in S$ has some attributes $\boldsymbol{x} = (x_1, \ldots, x_n)$ and a label ($l \in \{1, 2, \ldots, C\}$) associated with it. The entropy $H(S)$ is defined as

$$H(S) = -\sum_{l=1}^{C} p_{l,S} \log_2 p_{l,S}$$

$p_{l,S}$ denotes the fraction/probability of elements $s \in S$ that have label l.

Entropy The entropy defined above is conceptually very similar to that in physics and chemistry. Remember this (from JEE days)?

$$\Delta S_{\text{mix}} = -R(n_1 + n_2)\left(\frac{n_1}{n_1+n_2}\ln\frac{n_1}{n_1+n_2} + \frac{n_2}{n_1+n_2}\ln\frac{n_2}{n_1+n_2}\right)$$

Its the entropy of mixing 2 gases (n_1 and n_2 mols). Apart from the multiplicative constants, the expression is the same as $H(S)$.

Now we choose an attribute a from one of $x_1, \ldots, x_n$. Suppose a can attain values from Values(a). All the elements of $s \in S$ which have γ as the value of attribute a will be placed in S_γ. Now, the information gain is defined as

$$\text{Gain}(S, a) = H(S) - \sum_{\gamma \in \text{Values}(a)} \frac{|S_\gamma|}{|S|} H(S_\gamma)$$

Intuition : suppose we choose an ideal attribute that separates all the labels perfectly, then each of $H(S_\gamma)$ will be zero and information gain will be maximum. To gain information we need to reduce entropy by separating the elements. The factor $\frac{|S_\gamma|}{|S|}$ is needed to account for the fact that the set sizes are different. The attribute that gives the maximum information gain is the best attribute to split.

$$a^* = \arg\max_a \text{Gain}(S, a)$$

Apart from information gain there are other heuristics like **Gini impurity** that are commonly used in DTs.

Information Theory Basis The information gain that we defined above is called as mutual information in information theory.

$$I(X, Y) = H(Y) - H(Y|X) = H(X) - H(X|Y)$$

$H(X|Y)$ is the conditional entropy.

$$H(X|Y) = \sum_i p(X = x_i) H(Y|X = x_i)$$

Example Consider a dataset (Table 1) with two boolean attributes x_1 and x_2 and label y. Let us find the optimal attribute to split.

x_1	x_2	y
1	1	1
1	0	1
1	1	1
1	0	1
0	1	1
0	0	0

Table 1: Dataset for the example

$$H(S) = -\left(\frac{5}{6}\log_2\frac{5}{6} + \frac{1}{6}\log_2\frac{1}{6}\right) = 0.65$$

$$\text{Gain}(S, x_1) = H(S) - \left(\frac{4}{6} \times 0 + \frac{2}{6} \times 1\right) = 0.32$$

$$\text{Gain}(S, x_2) = H(S) - \left(\frac{3}{6} \times 0 + \frac{3}{6} \times 0.92\right) = 0.19$$

Since information gain is maximum when split using x_1, it is best to split using x_1.

Till now we have considered the attributes of dataset to be discrete variables. What if they are continuous? In case of discrete attribute we can easily split using the different values of the attribute. For **continuous** attributes, we need to identify thresholds (τ) for the attribute values that we can use to design binary questions of the kind, "Is attribute $a \leq \tau$?" with yes/no answers. Here is the procedure to identify these thresholds:

- Let $v_1, \ldots, v_n$ be the values of attribute a in the training set.
- Sort the values in increasing order : $v_{i_1}, \ldots, v_{i_n}$.
- Midpoint m_j is defined as
$$m_j = \frac{v_{i_j} + v_{i_{j+1}}}{2}$$
- Choose all such midpoints m_j whose underlying instances on either side have different labels. These midpoints will serve as thresholds for this continuous attribute.

For example, consider an attribute a that takes the values $\{10, 26, 40, 50, 100, 120\}$ across six training instances with corresponding labels $\{0, 0, 0, 1, 1, 0\}$. The midpoints are $\{18, 33, 45, 75, 110\}$, and the ones that will get picked up as thresholds are $\{45, 110\}$ since they are midpoints located between two instances with differing labels. Thus, the questions that can be asked relevant to attribute a are "Is $a \leq 45$?" and "Is $a > 110$?".

Example: Consider the dataset in Table 2 with two attributes x_1, x_2 which are real numbers. Each instance of the dataset has an associated binary label y (0 or 1).

x_1	x_2	y
0.1	0.53	1
0.2	0.86	1
0.25	0.36	0
0.36	0.91	1
0.47	0.87	1
0.65	0.13	0
0.71	0.82	0
0.85	0.55	0

Table 2: Dataset for the example

Let us start by splitting using x_1. (See the figure below)

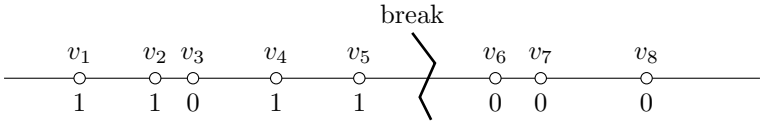


Figure 10: Visualisation of attribute x_1 of Table 2

To maximise information gain, we split between v_5 and v_6 using

$$m_5 = \frac{v_5 + v_6}{2} = 0.56$$

4 Stopping Criteria

Now we come to the second question, when do we stop splitting? These are some of the stopping criteria that help avoid overfitting to the data.

- Stop splitting if all the instances have the same label.
- Do not split if the number of instances at a node is less than k (k is some constant).
- Do not split if the number of leaf nodes exceed a certain threshold.
- Do not split if information gain at a node is below a certain threshold.

The above criteria act as some form of regularization for the DTs. Limiting the max depth of the tree is another possible regularization.

5 Pruning

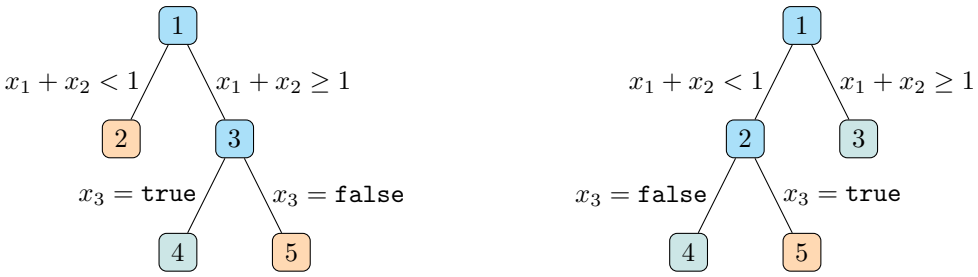
This is an alternate approach to tackle overfitting. Here we build the complete tree and then remove/prune some of the non-critical/redundant branches. To evaluate the performance after pruning, validation set is used.
Pruning a node means replacing the subtree rooted at the node with a leaf node whose label is the max label of the node that was pruned. We can use a greedy approach to pruning:

1. Pick a node which when pruned results in the highest gain in validation accuracy.
2. Repeat step 1 until there is no further improvement in validation accuracy.

This is called as **reduced error pruning**.

6 Random Forests (Brief Introduction)

Decision trees in the form that we studied till now, are very rarely used today. However, DTs are used in random forests (RTs), which is a very common method in **ensemble** learning. An ensemble method uses a combination of methods/models to predict the final output. A random forest is an ensemble model which uses **bootstrap aggregation** or **bagging**.
Bagging involves training an ensemble on bootstrapped data sets. Consider a dataset $\mathcal{D}$ having n samples in it (that is $|\mathcal{D}| = n$. Now we generate m datasets $\mathcal{D}_1, \ldots, \mathcal{D}_m$ each of the same size n ($|\mathcal{D}_1| = \cdots = |\mathcal{D}_m| = n$) by randomly sampling (with replacement) from $\mathcal{D}$. As a result an instance $d \in \mathcal{D}$ might occur zero, once or more than once in $\mathcal{D}_i$. Each of the m datasets is used in training a separate model. The final prediction is made by **aggregating** the results of each of the models (example averaging or voting).
Random forests (Figure 3) are bagged DTs, each of the m datasets is made into a separate DT. In each of these DTs, we pick a random set S (not too small not too large) of attributes to split the data. Taking very small set of attributes will result in losing too much information and taking too many will result in all the m models being nearly the same.



(a) Tree 1 (b) Tree 2

Figure 11: A random forest with $m = 2$