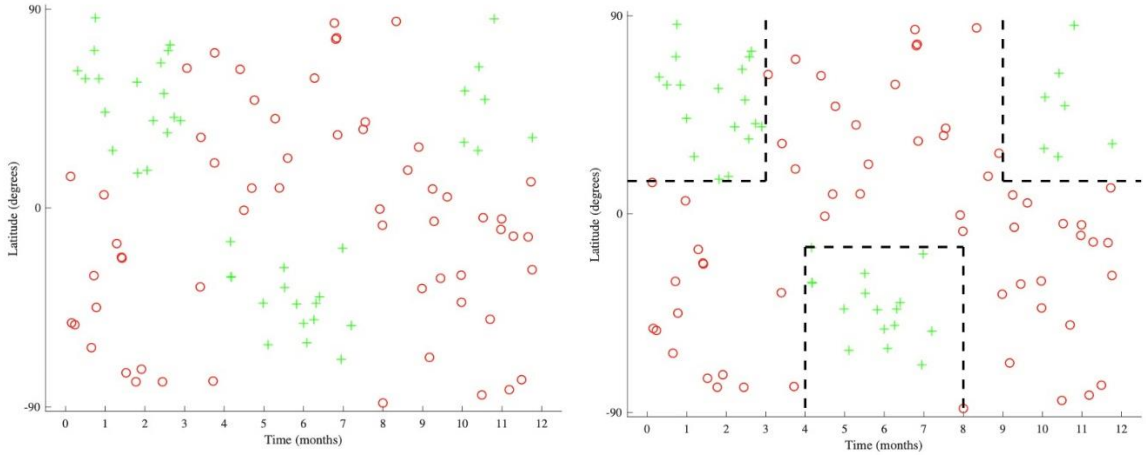
Decision tree

1 Overview

- **Decision Tree** is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.
- **Decision tree: supervised learning**
- **Tasks:** classification and regression
- **Learning methods**: constructed a tree by asking a set of Yes/No questions by choosing suitable features and thresholds.
- **Prediction**: given a data sample, starting from the root, put the data sample into one leaf node by checking a set of Yes/No questions.

2 Non-linear

- Decision trees can directly produce non-linear hypothesis functions without the need for first coming up with an appropriate feature mapping $\phi(x)$, which is adopted in SVM kernels.
- **Example**: let us say we want to build a classifier that, given a time and a location, can predict whether or not it would be possible to ski nearby. The time is represented as month of the year and the location is represented as a latitude (how far North or South we are with −90◦,0◦, and 90◦ being the South Pole, Equator, and North Pole, respectively).



- **Key idea**: Even though there is no linear boundary that can split the dataset, we can still recognize different areas of positive and negative space. The key idea is to partition the input space X into disjoint subsets (or regions) R_i.

$$\mathcal{X} = \bigcup_{i=0}^{n} R_i$$
$$\text{s.t.} \quad R_i \cap R_j = \emptyset \text{ for } i \neq j$$

Where $n \in Z^+$.

3 Selecting regions

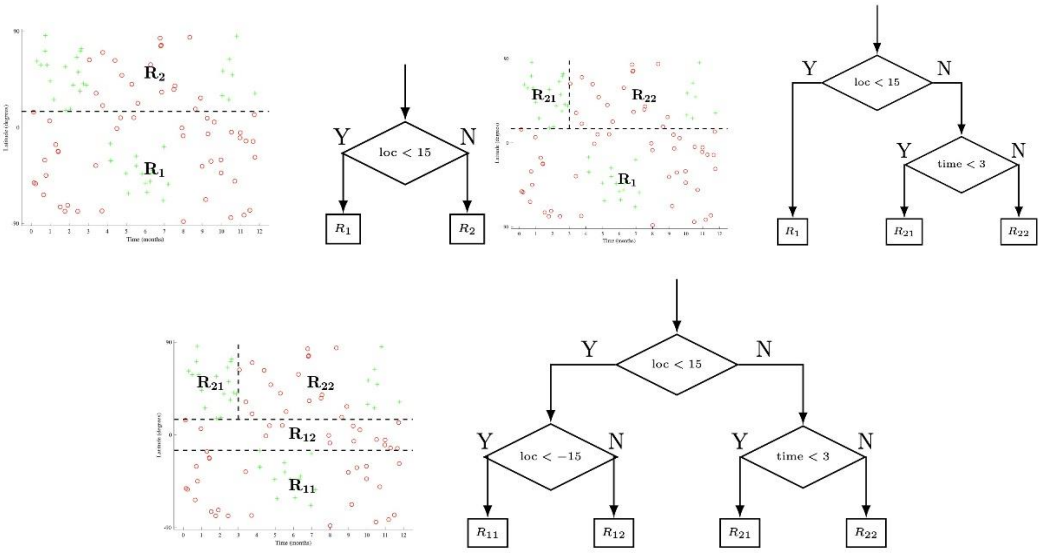
- Decision trees: generate an approximate solution via greedy, top-down, recursive partitioning.
- Why top-down?
  a) Start with the original input space X and split it into two child regions by thresholding on a single feature.
  b) Always selecting a leaf node, a feature, and a threshold to form a new split and repeat a) until reaching our goal.

manner, always selecting a leaf node, a feature, and a threshold to form a new split. Formally, given a parent region R_p, a feature index j, and a threshold $t \in \mathbb{R}$, we obtain two child regions R_1 and R_2 as follows:

$$R_1 = \{X \mid X_j < t, X \in R_p\}$$
$$R_2 = \{X \mid X_j \geq t, X \in R_p\}$$

- Example (continued): split dataset based on different features and thresholds.

The beginning of one such process is shown below applied to the skiing dataset. In step a, we split the input space $\mathcal{X}$ by the location feature, with a threshold of 15, creating child regions R_1 and R_2. In step b, we then recursively select one of these child regions (in this case R_2) and select a feature (time) and threshold (3), generating two more child regions (R_{21}). and (R_{22}). In step c, we select any one of the remaining leaf nodes (R_1, R_{21}, R_{22}). We can continue in such a manner until we a meet a given stop criterion (more on this later), and then predict the majority class at each leaf node.



- **Question**: How to decide the order of the features and their thresholds?

4 Define a loss function for features, threshold

- **Motivation**: choose suitable splits (without considering labels y_i)
- **Splitting**:
  a) Given a region R, let $L(R)$ denote the loss function of the set R.
  b) suppose we want to split a parent region R_p as two regions R_1 and R_2.

c) Within our greedy partitioning framework, we want to select the leaf region, feature, and threshold that will maximize our decrease in loss:

$$L(R_p) - \frac{|R_1| L(R_1) + |R_2| L(R_2)}{|R_1| + |R_2|}$$

- **Loss function**: **Misclassification problem**: $L_{misclass}$ represents the number of examples that would be misclassified if we predicted the majority class for region R.

> • For a classification problem, we are interested in the misclassification loss L_{misclass}. For a region R let $\hat{p}_c$ be the proportion of examples in R that are of class c. Misclassification loss on R can be written as:
>
> $$L_{\text{misclass}}(R) = 1 - \max_c(\hat{p}_c)$$

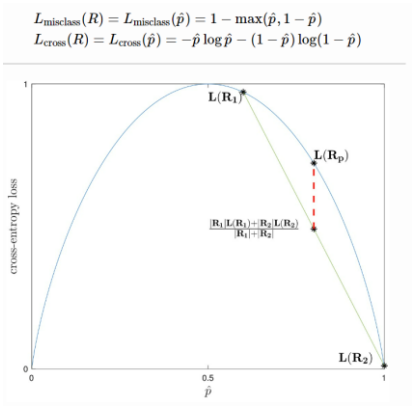
However, it is not very sensitive to changes in class probabilities.

- **Loss function**: cross-entropy loss L_{cross} is (similar to entropy?)

$$L_{cross}(R) = -\sum_c \hat{p}_c \log_2 \hat{p}_c$$

Where the reduction in loss from parent to child is known as information gain.

- Comparison of two loss functions:



> In the figure above on the left, we see the cross-entropy loss plotted over p. We take the regions (R_p, R_1, R_2) from the previous page's example's first split, and plot their losses as well. As cross-entropy loss is strictly concave, it can be seen from the plot (and easily proven) that as long as $\hat{p}_1 \neq \hat{p}_2$ and both child regions are non-empty, then the weighted sum of the children losses will always be less than that of the parent.
>
> Misclassification loss, on the other hand, is not strictly concave, and therefore there is no guarantee that the weighted sum of the children will be less than that of the parent, as shown above right, with the same partition. Due to this added sensitivity, cross-entropy loss (or the closely related Gini loss) are used when growing decision trees for classification.

5 A loss function for regression

- Motivation: For each data sample x_i, suppose it has a label $y_i \in R$ that we want to predict. We want to estimate loss based on the prediction

- **Prediction**: Much of the tree growth process remains the same, with the difference being that the final prediction for a region R is the mean of all the values:

$$\hat{y} = \frac{\sum_{i \in R} y_i}{|R|}$$

- **Loss function**: We can directly use the squared loss to select our splits:

$$L_{\text{squared}}(R) = \frac{\sum_{i \in R} (y_i - \hat{y})^2}{|R|}$$

6 Categorical Variables

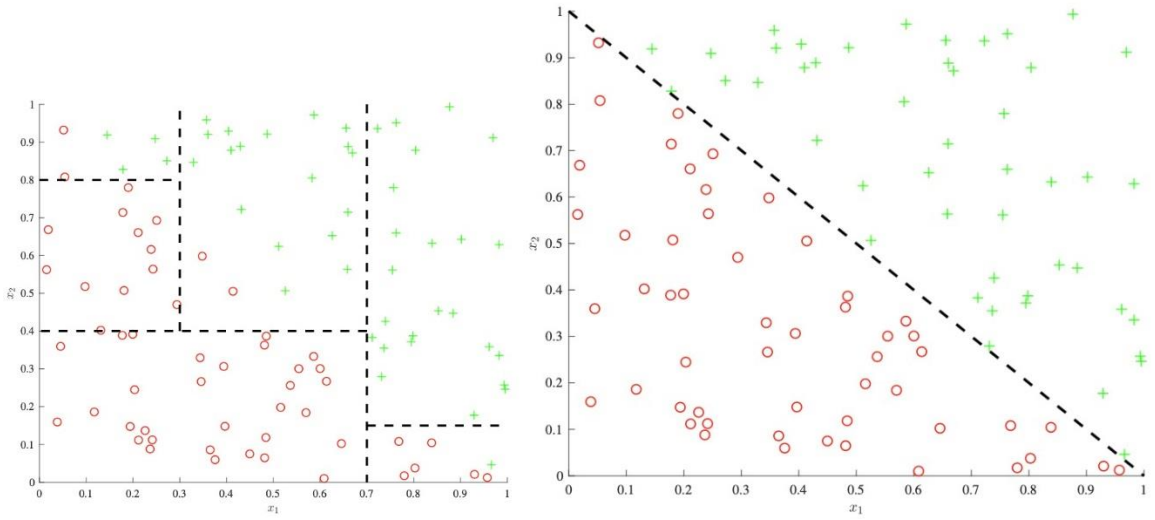
- **Advantages**: decision trees can easily deal with categorical variables.

7 Run time

- **Tree structure**: consider binary classification with n examples, f features, and a tree of depth d.
- **Training time**: since each sample can only appear in at most $O(d)$ nodes. Through sorting and intelligent caching of intermediate values, we can achieve runtime of O(1) at each node for a single data point for a single feature. Thus, overall runtime is $O(nfd)$– a fairly fast runtime as the data matrix alone is of size nf.
- **Testing time**: a data point traverses the tree until we reach a leaf node and then output its prediction, for a runtime of O(d).

8 Drawbacks

- Decision trees cannot easily capture additive structure.



9 Example: 2-class classification task

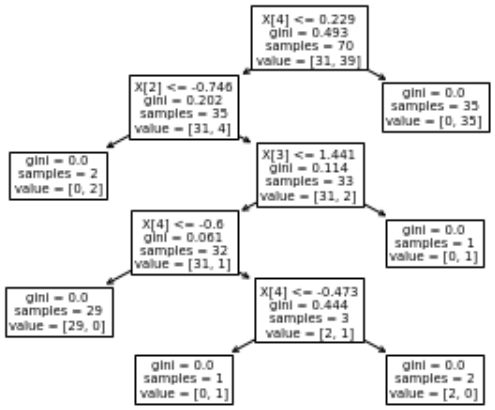
https://www.geeksforgeeks.org/decision-tree/

- **Gini Index**: a score that evaluates how accurate a split is among the classified groups. Gini index evaluates a score in the range between 0 and 1, where 0 is when all observations belong to one class, and 1 is a random distribution of the elements within classes. In this case, we want to have a Gini index score as low as possible. Gini Index is the evaluation metrics we shall use to evaluate our Decision Tree Model.
- **Sep 1**: Generate data with two classes

```
from sklearn.datasets import make_classification
from sklearn import tree
from sklearn.model_selection import train_test_split


'''
------------------------------------
Generate a n-class problem
'''

X, t = make_classification(100, 5, n_classes=2, shuffle=True, random_state=10)


'''
```

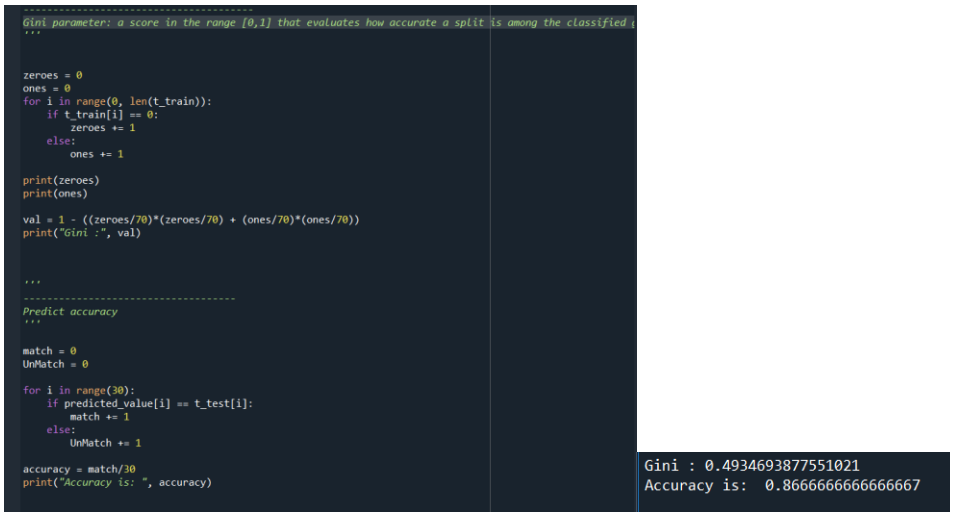
- **Step 2**: split the dataset into training and testing datasets, and train the model, predict test data.

```
-------------------------------------
train models and predict values
Shuffle=True: prevent correlation among data
'''
X_train, X_test, t_train, t_test = train_test_split(X, t, test_size=0.3, shuffle=True, random_state=1)

model = tree.DecisionTreeClassifier()
model = model.fit(X_train, t_train)


tree.plot_tree(model)


predicted_value = model.predict(X_test)
print(predicted_value)
```



The decision tree [may lead to overfit since some leaf nodes only contain one sample]

- **Step 3**: calculate **Gini Index** and accuracy



10 Tips for practical use

- Decision trees tend to overfit on data with a large number of features. Getting the right ratio of samples to number of features is important, since a tree with few samples in high dimensional space is very likely to overfit.
- Consider performing dimensionality reduction (PCA, ICA, or Feature selection) beforehand to give your tree a better chance of finding features that are discriminative.
- Understanding the decision tree structure will help in gaining more insights about how the decision tree makes predictions, which is important for understanding the important features in the data.
- Visualize your tree as you are training by using the export function. Use max_depth=3 as an initial tree depth to get a feel for how the tree is fitting to your data, and then increase the depth.
- Use `min_samples_split` or `min_samples_leaf` to ensure that multiple samples inform every decision in the tree, by controlling which splits will be considered. A very small number will usually mean the tree will overfit, whereas a large number will prevent the tree from learning the data. Try `min_samples_leaf=5` as an initial value.
- Decision tree does not perform very well for multiple-class tasks.

https://scikit-learn.org/stable/modules/tree.html

References

https://www.geeksforgeeks.org/decision-tree/

https://aman.ai/cs229/decision-trees/

https://www.coursera.org/articles/decision-tree-machine-learning

Cross Entropy vs K-L Divergence

$$S(v) = -\sum_i p(v_i) \log p(v_i),$$

Entropy: A random variable has information about itself - self-informativeness.

$$D_{KL}(A \parallel B) = \sum_i p_A(v_i) \log p_A(v_i) - p_A(v_i) \log p_B(v_i) = \sum_i p_A(v_i)[log(p_A(v_i) - p_B(v_i)]$$

True distribution — How B differs from A

$$H(A, B) = -\sum_i p_A(v_i) \log p_B(v_i).$$

Cross-Entropy: A random variable compares true distribution A with approximated distribution B.

Cross-entropy = divergence + entropy
[A random variable knows about itself (**entropy**) and **from its perspective** compares its true distribution with approximated distribution **through divergence**]
Minimizing divergence and cross-entropy are said to have the same effects.

Relative-Entropy: A random variable compares true distribution A with how the approximated distribution B differs from A at each sample point (divergence or difference).

$$H(A, B) = D_{KL}(A \parallel B) + S_A.$$