Lorraine Li Follow

May 16, 2019 · 8 min read ★ · Listen

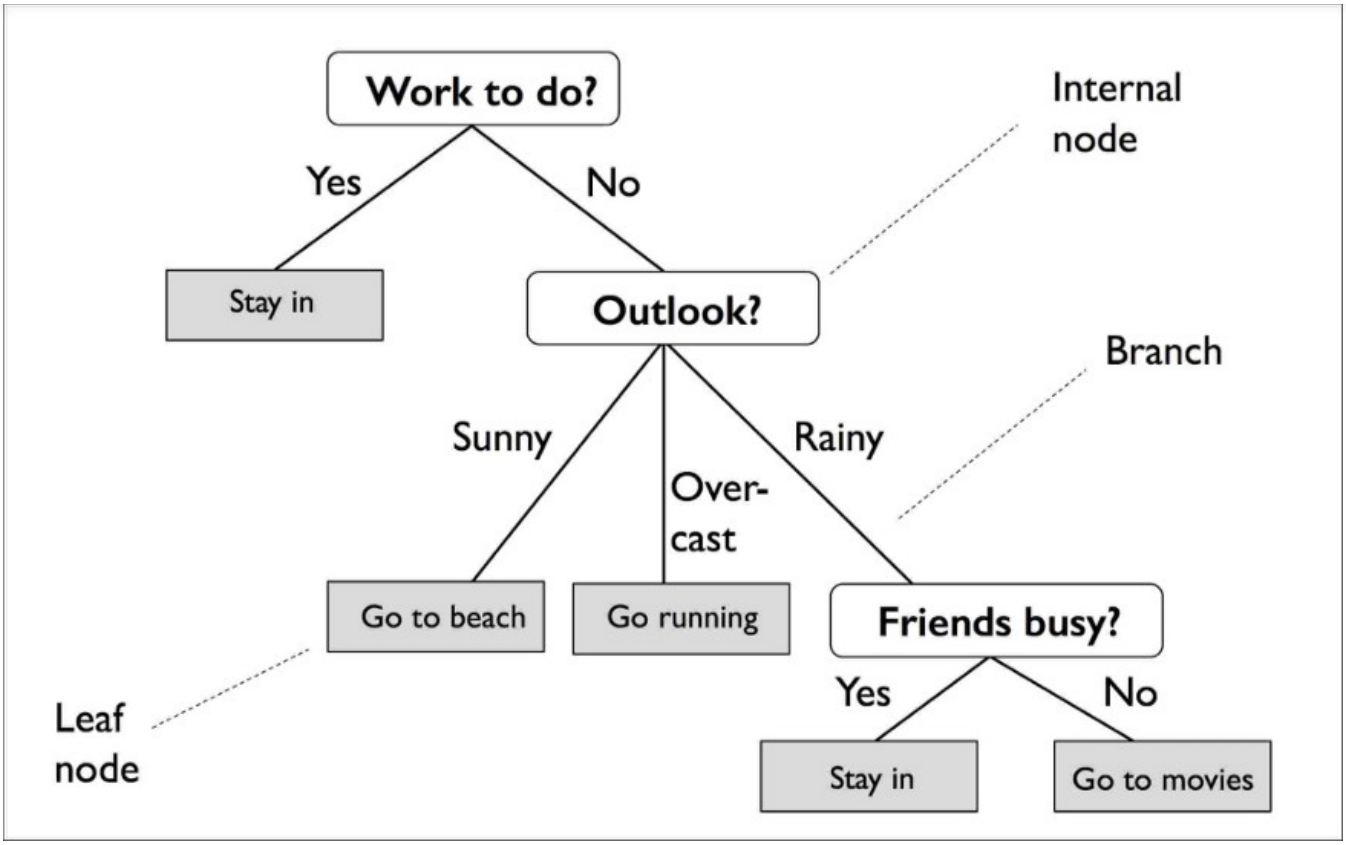
Save

Classification and Regression Analysis with Decision Trees

By understanding the fundamental concepts and mathematics behind decision trees, learn to build classification and regression decision trees!

A **decision tree** is a supervised machine learning model used to predict a target by learning decision rules from features. As the name suggests, we can think of this model as breaking down our data by making a decision based on asking a series of questions.

Let's consider the following example in which we use a decision tree to decide upon an activity on a particular day:



Based on the features in our training set, the decision tree model learns a series of questions to infer the class labels of the samples. As we can see, decision trees are attractive models if we care about interpretability.

Although the preceding figure illustrates the concept of a decision tree based on categorical targets (**classification**), the same concept applies if our targets are real numbers (**regression**).

In this tutorial, we will discuss how to build a decision tree model with Python's `scikit-learn` library. We will cover:

- The fundamental concepts of decision trees
- The mathematics behind the decision tree learning algorithm
- Information gain and impurity measures
- Classification trees
- Regression trees

> *This tutorial is adapted from Next Tech's **Python Machine Learning** series which takes you through machine learning and deep learning algorithms with Python from 0 to 100. It includes an in-browser sandboxed environment with all the necessary software and libraries pre-installed, and projects using public datasets. You can get started here!*

The Fundamentals of Decision Trees

A decision tree is constructed by **recursive partitioning** — starting from the root node (known as the first **parent**), each node can be split into left and right **child** nodes. These nodes can then be further split and they themselves become parent nodes of their resulting children nodes.

For example, looking at the image above, the root node is `Work to do?` and splits into the child nodes `Stay in` and `Outlook` based on whether or not there is work to do. The `Outlook` node further splits into three child nodes.

So, how do we know what the optimal splitting point is at each node?

Starting from the root, the data is split on the feature that results in the largest **Information Gain** (**IG**) (explained in more detail below). In an iterative process, we then repeat this splitting procedure at each **child node** until the leaves are pure — i.e. samples at each node all belong to the same class.

In practice, this can result in a very deep tree with many nodes, which can easily lead to overfitting. Thus, we typically want to **prune** the tree by setting a limit for the maximal depth of the tree.

Maximizing Information Gain

In order to split the nodes at the most informative features, we need to define an objective function that we want to optimize via the tree learning algorithm. Here, our objective function is to maximize the information gain at each split, which we define as follows:


Here, *f* is the feature to perform the split, *Dp*, *Dleft*, and *Dright* are the datasets of the parent and child nodes, *I* is the **impurity measure**, *Np* is the total number of samples at the parent node, and *Nleft* and *Nright* are the number of samples in the child nodes.

We will discuss impurity measures for classification and regression decision trees in more detail in our examples below. But for now, just understand that information gain is simply the difference between the impurity of the parent node and the sum of the child node impurities — the lower the impurity of the child nodes, the larger the information gain.

Note that the above equation is for binary decision trees — each parent node is split into two child nodes only. If you have a decision tree with multiple nodes, you would simply sum the impurity of all nodes.

Classification Trees

We will start by talking about classification decision trees (also known as **classification trees**). For this example, we will be using the *Iris* dataset, a classic in the field of machine learning. It contains the measurements of 150 *Iris* flowers from three different species — *Setosa*, *Versicolor*, and *Virginica*. These will be our **targets**. Our goal is to predict which category an *Iris* flower belongs to. The petal length and width in centimeters are stored as columns, which we also call the **features** of the dataset.

Let's first import the dataset and assign the features as `X` and the target as `y`:

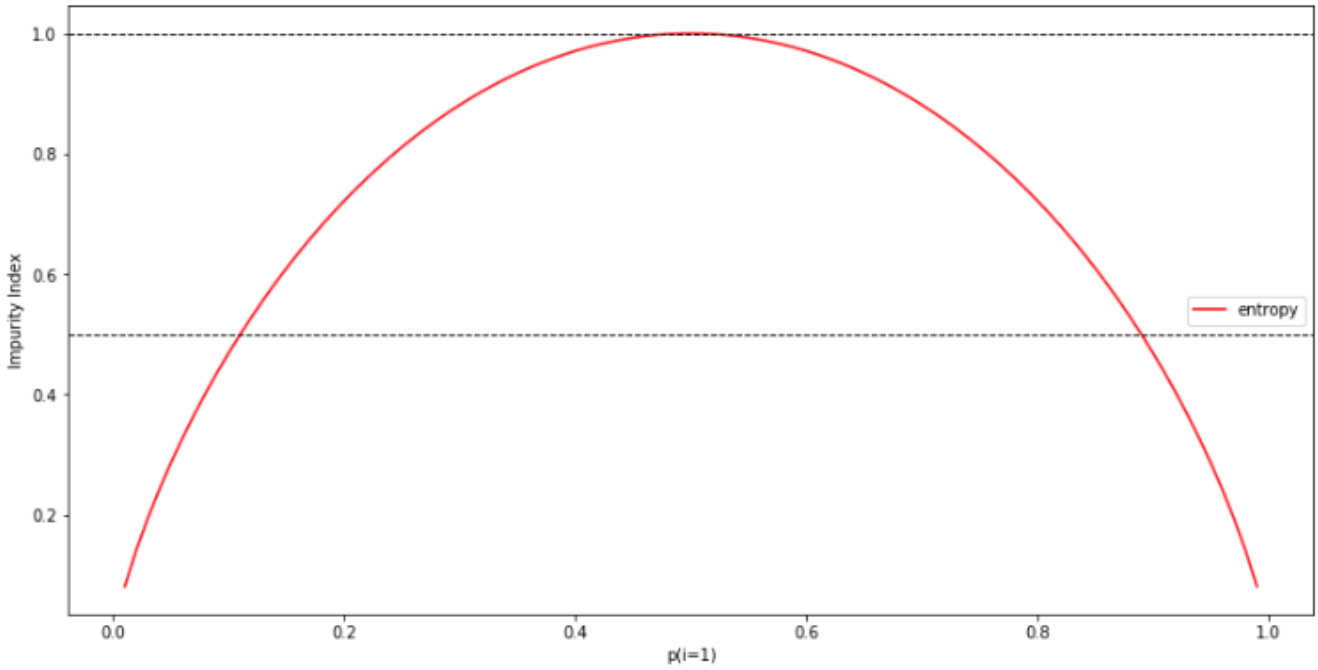

Using `scikit-learn`, we will now train a decision tree with a maximum depth of 4. The code is as follows:


Notice that we set the `criterion` as '**entropy**'. This criterion is known as the impurity measure (mentioned in the previous section). In classification, entropy is the most common impurity measure or splitting criteria. It is defined by:

$$E(t) = -\sum_{i=1}^{c} p(i|t) \log_2 p(i|t)$$

Here, $p(i|t)$ is the proportion of the samples that belong to class c for a particular node t. The entropy is therefore 0 if all samples at a node belong to the same class, and the entropy is maximal if we have a uniform class distribution.

For a more visual understanding of entropy, let's plot the impurity index for the probability range [0, 1] for class 1. The code is as follows:

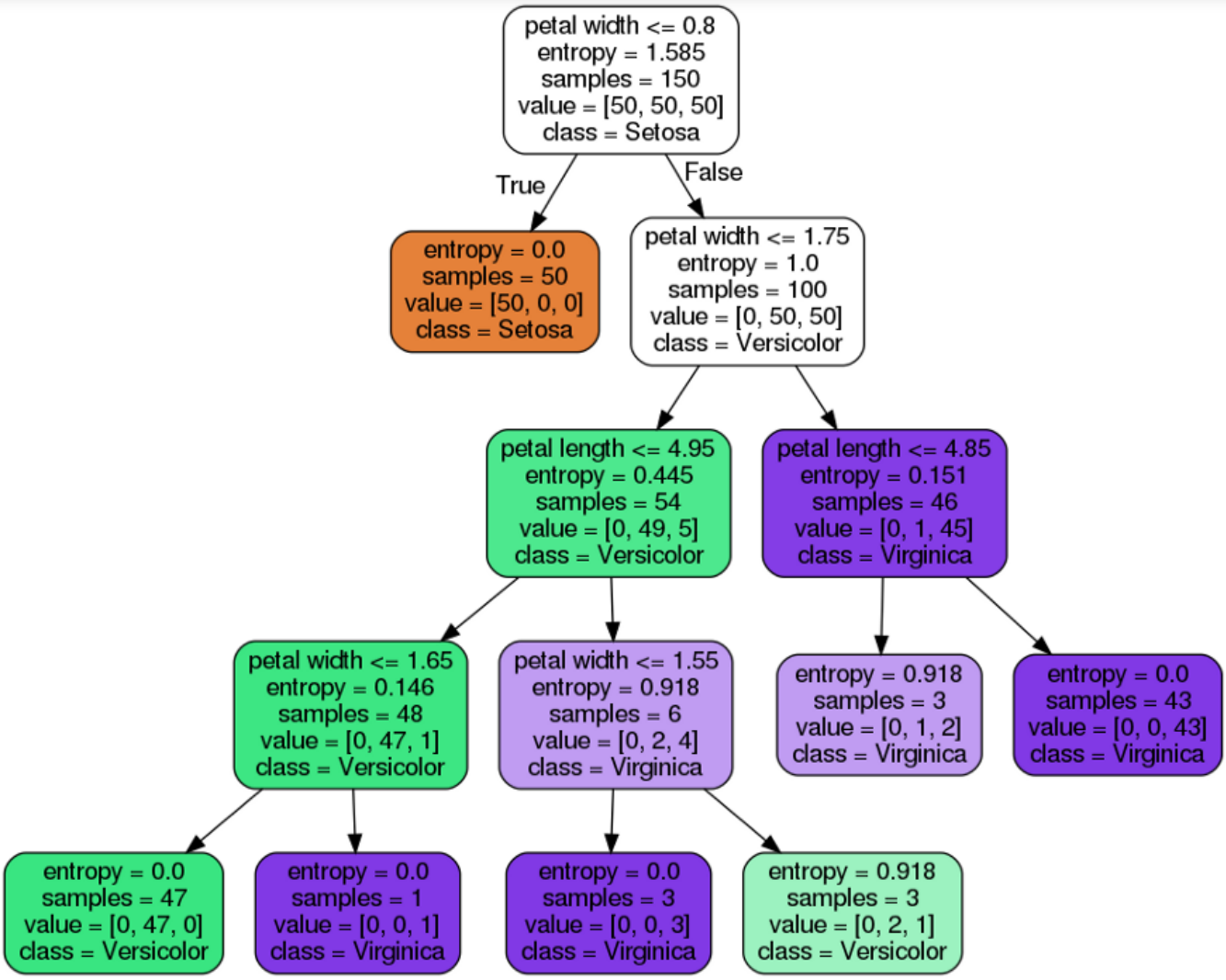
As you can see, entropy is 0 if $p(i=1|t) = 1$. If the classes are distributed uniformly with $p(i=1|t) = 0.5$, entropy is 1.

Now, returning to our *Iris* example, we will visualize our trained classification tree and see how entropy decides each split.

A nice feature in `scikit-learn` is that it allows us to export the decision tree as a `.dot` file after training, which we can visualize using GraphViz, for example. In addition to GraphViz, we will use a Python library called `pydotplus`, which has capabilities similar to GraphViz and allows us to convert `.dot` data files into a decision tree image file.

You can install `pydotplus` and `graphviz` by executing the following commands in your Terminal:

```
pip3 install pydotplus
apt install graphviz
```



tree.png

Looking at the resulting decision tree figure saved in the image file `tree.png`, we can now nicely trace back the splits that the decision tree determined from our training dataset. We started with 150 samples at the root and split them into two child nodes with 50 and 100 samples, using the **petal width** cut-off ≤ 1.75 cm. After the first split, we can see that the left child node is already pure and only contains samples from the `setosa` class (entropy = 0). The further splits on the right are then used to separate the samples from the `versicolor` and `virginica` class.

Looking at the final entropy we see that the decision tree with a depth of 4 does a very good job of separating the flower classes.

Regression Trees

$1000s) as the target and `LSTAT` (percentage of lower status of the population) as the feature.

Let's first import the necessary attributes from `scikit-learn` into a `pandas` DataFrame.

	LSTAT	MEDV
0	4.98	24.0
1	9.14	21.6

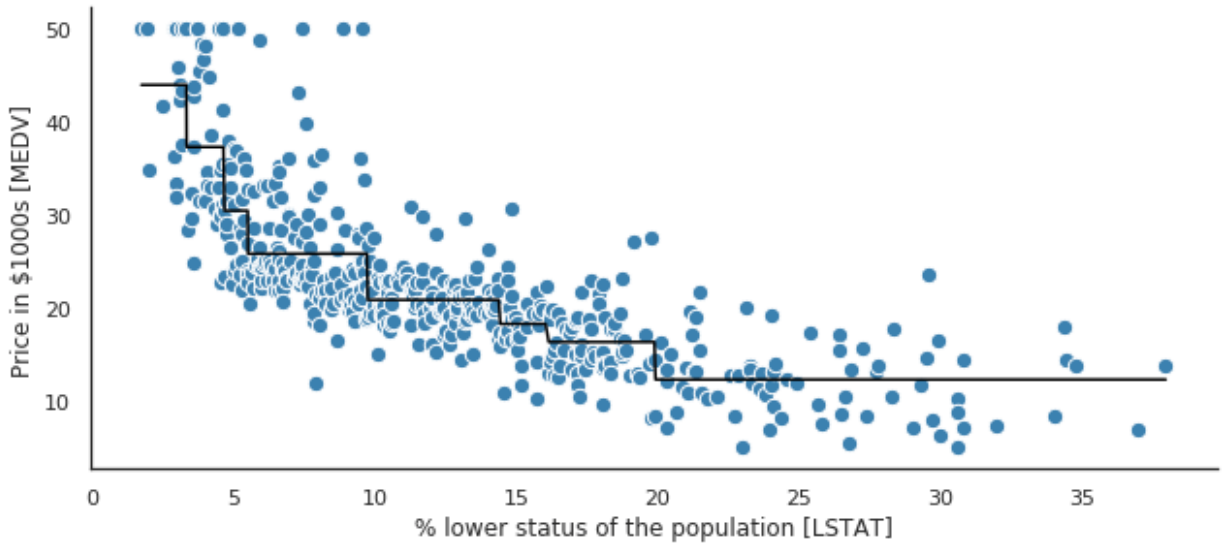

Let's use the `DecisionTreeRegressor` implemented in `scikit-learn` to train a regression tree:

Notice that our `criterion` is different from the one we used for our classification tree. Entropy as a measure of impurity is a useful criteria for classification. To use a decision tree for regression, however, we need an impurity metric that is suitable for continuous variables, so we define the impurity measure using the **weighted mean squared error** (**MSE**) of the children nodes instead:


Here, *Nt* is the number of training samples at node *t*, *Dt* is the training subset at node *t*, *y(i)* is the true target value, and *ŷt* is the predicted target value (sample mean):

$$\hat{y}_t = \frac{1}{N_t} \sum_{i \in D_t} y^{(i)}$$

Now, let's model the relationship between `MEDV` and `LSTAT` to see what the line fit of a regression tree looks like:

As we can see in the resulting plot, the decision tree of depth 3 captures the general trend in the data.

I hope you enjoyed this tutorial on decision trees! We discussed the fundamental concepts of decision trees, the algorithms for minimizing impurity, and how to build decision trees for both classification and regression.

In practice, it is important to know how to choose an appropriate value for a depth of a tree to not overfit or underfit the data. Knowing how to combine decision trees to form an ensemble **random forest** is also useful as it usually has a better generalization performance than an individual decision tree due to randomness, which helps to decrease the model's variance. It is also less sensitive to outliers in the dataset and doesn't require much parameter tuning.

*We cover these techniques in our **Python Machine Learning** series, as well as diving into other machine learning models such as perceptrons, Adaline, linear and polynomial regression, logistic regression, SVMs, kernel SVMs, k-nearest-neighbors, models for sentiment analysis, k-means clustering, DBSCAN, convolutional neural networks, and recurrent neural networks.*

You can get started here!