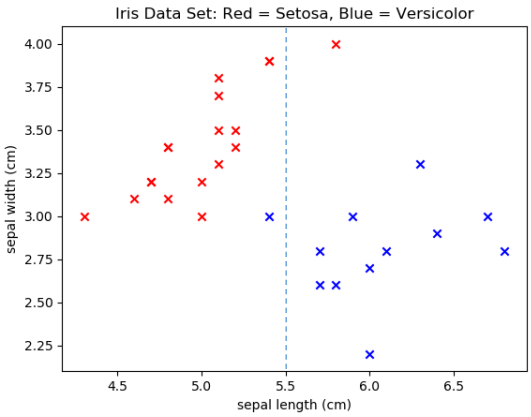
Decision Tree Classification

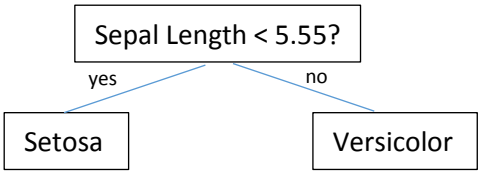
The Basic Idea

Decision Tree classifiers (sometimes referred to as CART – Classification and Regression Trees) repeatedly divide the training data into two portions by finding a good cut point on a single feature.

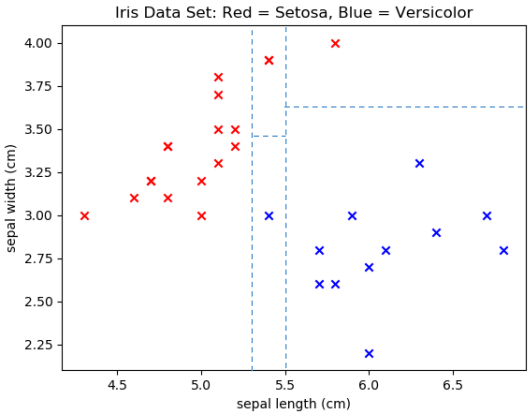
The scatterplot on the right shows 30 data points from the Iris Data Set found at the UCI Machine Learning Repository, graphed for Sepal Length (x-axis) and Sepal Width (y-axis). The raw data is in a separate handout.

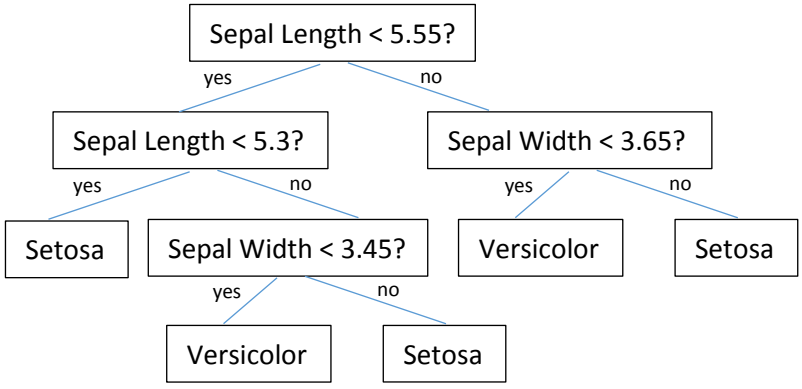


The first cut point chosen by a Decision Tree Classifier might be sepal length = 5.55, as this divides the data into two relatively pure groups. That leads to the following decision tree:



This tree mis-classifies 2 of the 30 examples. After this first cut, the process repeats (recursively) on the right side and the left side, possibly leading to the following tree:





This Decision Tree achieves 100% accuracy on the training set. How will it perform on the testing set?

Decision Tree algorithms are often implemented as **greedy** algorithms (they always take the best looking split at each step, and never backtrack). They are also **divide and conquer** algorithms (they repeatedly divide big problems into smaller problems), and they have a naturally **recursive** implementation.

Decision Tree Pros and Cons

Here are a few of the pros and cons of decision trees vs. other learning algorithms…

Pro: Interpretability

Decision trees can be easily read and understood by human beings. This makes them attractive, especially in domains such as medicine and the law where it is important not only to make good decisions, but to be able to understand and justify them.

Pro: Feature Selection

Once constructed, decision trees tend to select only a few features as relevant, whereas kNN and other algorithms always make use of all the features in the data. If you have irrelevant features, kNN might get thrown off by them, but decision trees will automatically ignore them.

Pro: No Need to Normalize the data

Since decision trees don't rely on any distance measurement (such as kNN), scaling or normalizing the features is not necessary.

Con: High Error Rate on Small Data Sets

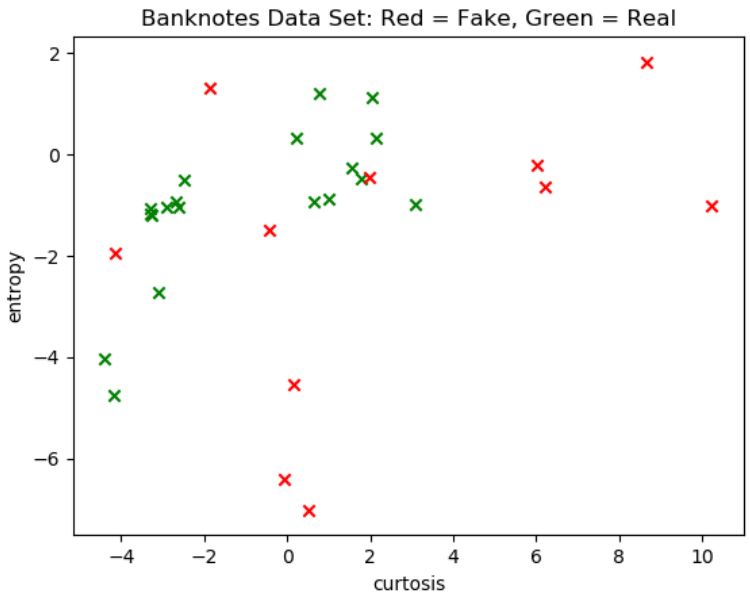
If you have a small training set, making vertical and horizontal cuts through the feature space might not be a good match for the natural distribution of the data, and can lead to high error rates.

Con: Exponential Growth

As the problem sets get bigger (more data, more features) decision tree algorithms tend to get exponentially slower.

Decision Tree Learning Unplugged

In a group of 2 or 3, try and apply the decision tree algorithm above to split the Banknotes data shown in the scatterplot on the right. The raw data is in a separate handout if you need it.



It's up to you to decide the "best" cut point at each step but try to be consistent. It's also up to you to decide when to stop splitting. Will you go for 100% accuracy on the training set, or can you tolerate some error?

Debrief: Finding the Best Split

How did you determine the "best" split in the activity above? Did you simply count the "out of place" items on each possible split and tried to minimize it? Or did you do something different?

Actual implementations of decision tree learners use an equation to figure out how good each split is. In most cases, they use either the **Gini Impurity** or **Information Gain**, both of which are supported in sklearn. Roughly, Gini Impurity is a measure of the probability of a randomly chosen element being misclassified if each split is adopted. When a split is pure, the Gini Impurity is 0. Information Gain is a measure of how much each split reduces the amount of information needed to correctly classify the items. Both measures tend to come out better for splits that are more pure.

On a large data set with lots of features, finding the best split can be time consuming. In cases like this, it might be better to try randomly selecting features and cut points and use the best of the ones you find. You may not end up with the optimal tree, but you should end up with a pretty good one.

Debrief: Stopping Criteria

How did you decide when to stop splitting in the activity above?

In the sklearn implementation, the default is to keep going until every training item is correctly classified. But this can lead to **overfitting**, especially with large and noisy data sets. It can also lead to huge and dense trees that are harder for a human to understand.

Common ways to stop a decision tree getting too detailed include setting a maximum depth or a maximum allowed number of leaf nodes for the tree. (The depth of a tree is the maximum number of steps from the root to a leaf.) Another common way is to set a minimum number of data points in a leaf or a minimum number of data points that can be split. SKLearn will allow you to experiment with all these stopping criteria.

Solution & Data

For the raw data and a sample solution, generated by the Decision Tree algorithm in sklearn and graphed using webgraphviz.com, see the accompanying document.

Exercises

1. Go to decision_trees_example.py and modify the code so that it runs the algorithm 10 times with a different testing and training split each time. Report the average accuracy over 10 runs.

2. In SKLearn, use a new data set of your choice to experiment with different parameters for decision tree generation (criterion, splitter) and stopping condition (max_depth, min_samples_split, min_samples_leaf). Which combination gives you the highest accuracy? Which of these parameters has the biggest impact on the accuracy and the shape of the tree? What's the best balance of accuracy and tree readability?

3. There are lots of other classifiers in sklearn and they all work in a similar way to decision trees and nearest neighbor. Pick one, read about it, and implement it on a data set of your choice.