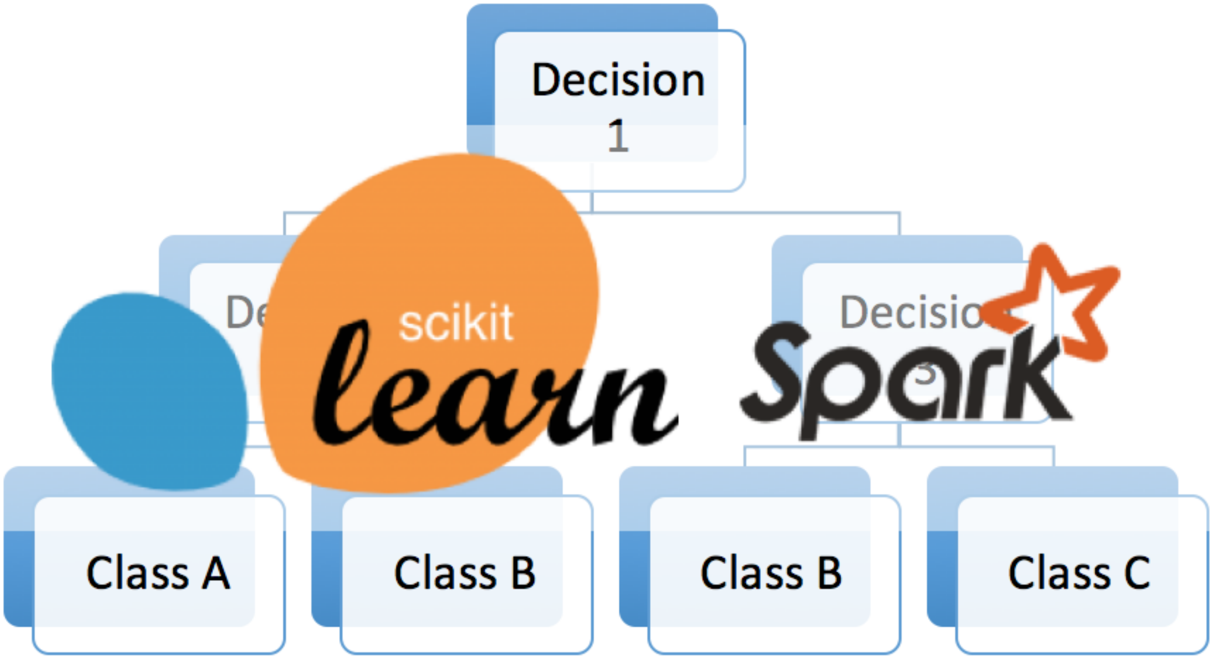
The Mathematics of Decision Trees, Random Forest and Feature Importance in Scikit-learn and Spark

Stacey Ronaghan Follow

May 12, 2018 · 6 min read



Introduction

This post attempts to consolidate information on tree algorithms and their implementations in Scikit-learn and Spark. In particular, it was written to provide clarification on how feature importance is calculated.

There are many great resources online discussing how decision trees and random forests are created and this post is not intended to be that. Although it includes short definitions for context, it assumes the reader has a grasp on these concepts and wishes to know how the algorithms are implemented in Scikit-learn and Spark.

So, let's start with…

Decision Trees

Decision trees learn how to best split the dataset into smaller and smaller subsets to predict the target value. The condition, or test, is represented as the "leaf" (node) and the possible outcomes as "branches" (edges). This splitting process continues until no further gain can be made or a preset rule is met, e.g. the maximum depth of the tree is reached.

Decision Tree Algorithms

There are multiple algorithms and the scikit-learn documentation provides an overview of a few of these (link)

So what do Scikit-learn and Spark use?

Scikit-learn documentation states it is using "an optimized version of the CART algorithm". Whilst not explicitly mentioned in the documentation, it has been inferred that Spark is using ID3 with CART.

So let's focus on these two — ID3 and CART.

The advantages and disadvantages have been taken from the paper Comparative Study ID3, CART and C4.5 Decision Tree Algorithm: A Survey. More thorough definitions can also be found there.

ID3

The algorithm creates a multi-way tree — each node can have two or more edges — finding the categorical feature that will maximize the *information gain* using the impurity criterion *entropy*. Not only can it not handle numerical features, it is only appropriate for classification problems.

Advantages

- Understandable prediction rules are created from the training data
- Builds the fastest tree
- Builds a short tree
- Only need enough attributes until all data is classified
- Finding leaf nodes enable test data to be pruned, reducing number of tests
- Whole dataset is searched to create tree

Disadvantages

- Data may be over-fitted or over-classified, if a small sample is tested
- Only one attribute at a time is tested for making a decision
- Does not handle numeric attributes and missing values

CART

CART stands for Classification and Regression Trees. The algorithm creates a binary tree — each node has exactly two outgoing edges — finding the best numerical or categorical feature to split using an appropriate impurity criterion. For classification, *Gini impurity* or *twoing criterion* can be used. For regression, CART introduced *variance reduction* using least squares (mean square error).

Advantages

- CART can easily handle both numerical and categorical variables
- CART algorithm will itself identify the most significant variables and eliminate non-significant ones
- CART can easily handle outliers

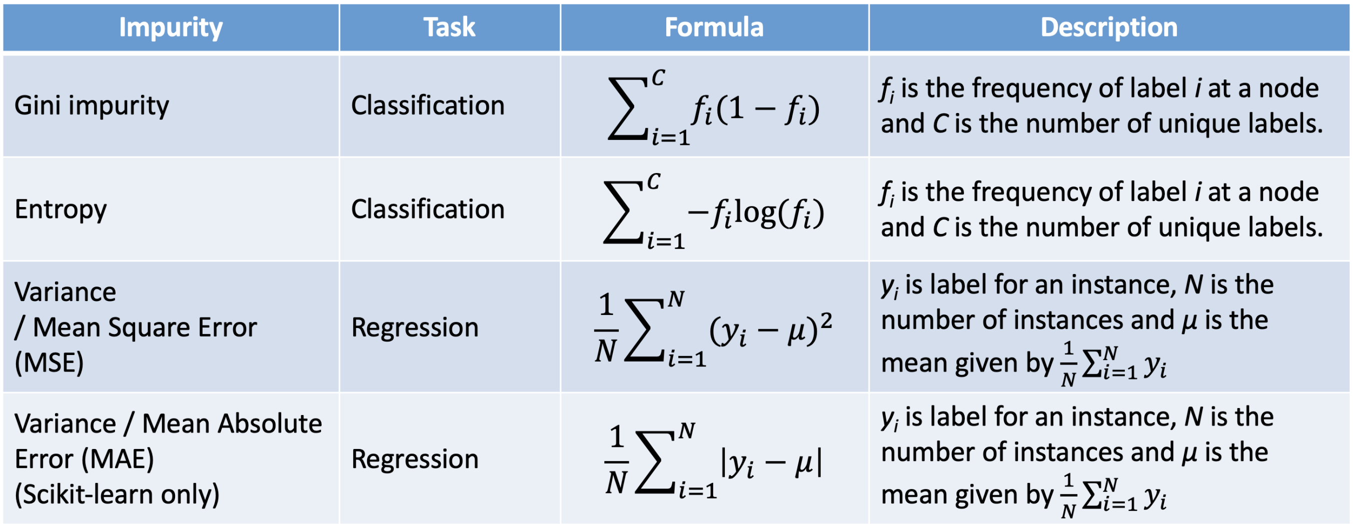
Disadvantages

- CART may have unstable decision tree

- CART splits by one by one variable

Node impurity / Impurity Criterion

Both Scikit-learn and Spark provide information in their documentation on the formulas used for impurity criterion. For classification, they both use Gini impurity by default but offer Entropy as an alternative. For regression, both calculate variance reduction using Mean Square Error. Additionally, variance reduction can be calculated with Mean Absolute Error in Scikit-learn.



Impurity	Task	Formula	Description
Gini impurity	Classification	$\sum_{i=1}^{C} f_i(1-f_i)$	f_i is the frequency of label i at a node and C is the number of unique labels.
Entropy	Classification	$\sum_{i=1}^{C} -f_i\log(f_i)$	f_i is the frequency of label i at a node and C is the number of unique labels.
Variance / Mean Square Error (MSE)	Regression	$\frac{1}{N}\sum_{i=1}^{N}(y_i-\mu)^2$	y_i is label for an instance, N is the number of instances and μ is the mean given by $\frac{1}{N}\sum_{i=1}^{N} y_i$
Variance / Mean Absolute Error (MAE) (Scikit-learn only)	Regression	$\frac{1}{N}\sum_{i=1}^{N}\lvert y_i-\mu\rvert$	y_i is label for an instance, N is the number of instances and μ is the mean given by $\frac{1}{N}\sum_{i=1}^{N} y_i$

Impurity Formulas used by Scikit-learn and Spark

Links to Documentation on Tree Algorithms

- Sci-kit learn
- Spark

Information Gain

Another term worth noting is “Information Gain” which is used with splitting the data using entropy. It is calculated as the decrease in entropy after the dataset is split on an attribute:

> *Gain(T,X) = Entropy(T) — Entropy(T,X)*

- T = target variable

- X = Feature to be split on
- Entropy(T,X) = The entropy calculated after the data is split on feature X

Random Forests

Random forests (RF) construct many individual decision trees at training. Predictions from all trees are pooled to make the final prediction; the mode of the classes for classification or the mean prediction for regression. As they use a collection of results to make a final decision, they are referred to as Ensemble techniques.

Feature Importance

Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The node probability can be calculated by the number of samples that reach the node, divided by the total number of samples. *The higher the value the more important the feature.*

Implementation in Scikit-learn

For each decision tree, Scikit-learn calculates a nodes importance using Gini Importance, assuming only two child nodes (binary tree):

$$ni_j = w_j C_j - \ w_{left(j)} C_{left(j)} - \ w_{right(j)} C_{right(j)}$$

- ni sub(j)= the importance of node j
- w sub(j) = weighted number of samples reaching node j
- C sub(j)= the impurity value of node j
- left(j) = child node from left split on node j
- right(j) = child node from right split on node j

sub() is being used as subscript isn't available in Medium

See method *compute_feature_importances* in _tree.pyx

The importance for each feature on a decision tree is then calculated as:

$$fi_i = \frac{\sum_{j:node\ j\ splits\ on\ feature\ i} ni_j}{\sum_{k \in all\ nodes} ni_k}$$

- fi sub(i)= the importance of feature i
- ni sub(j)= the importance of node j

These can then be normalized to a value between 0 and 1 by dividing by the sum of all feature importance values:

$$normfi_i = \frac{fi_i}{\sum_{j \in all\ features} fi_j}$$

The final feature importance, at the Random Forest level, is it's average over all the trees. The sum of the feature's importance value on each trees is calculated and divided by the total number of trees:

$$RFfi_i = \frac{\sum_{j \in all\ trees} normfi_{ij}}{T}$$

- RFfi sub(i)= the importance of feature i calculated from all trees in the Random Forest model
- normfi sub(ij)= the normalized feature importance for i in tree j
- T = total number of trees

See method *feature_importances_* in forest.py

Notation was inspired by this StackExchange thread which I found incredible useful for this post.

Implementation in Spark

For each decision tree, Spark calculates a feature's importance by summing the gain, scaled by the number of samples passing through the node:

$$fi_i = \sum_{j:nodes\ j\ splits\ on\ feature\ i} s_j C_j$$

- fi sub(i) = the importance of feature i
- s sub(j) = number of samples reaching node j
- C sub(j) = the impurity value of node j

See method *computeFeatureImportance* in treeModels.scala

To calculate the final feature importance at the Random Forest level, first the feature importance for each tree is normalized in relation to the tree:

$$normfi_i = \frac{fi_i}{\sum_{j \in all\ features} fi_j}$$

- normfi sub(i) = the normalized importance of feature i
- fi sub(i) = the importance of feature i

Then feature importance values from each tree are summed normalized:

$$RFfi_i = \frac{\sum_j normfi_{ij}}{\sum_{j\in all\ features, k\in all\ trees} normfi_{jk}}$$

- RFfi sub(i)= the importance of feature i calculated from all trees in the Random Forest model
- normfi sub(ij)= the normalized feature importance for i in tree j

See method *featureImportances* in treeModels.scala

Conclusion

This goal of this model was to explain how Scikit-Learn and Spark implement Decision Trees and calculate Feature Importance values.

Hopefully by reaching the end of this post you have a better understanding of the appropriate decision tree algorithms and impurity criterion, as well as the formulas used to determine the importance of each feature in the model.

Machine Learning Scikit Learn Spark Random Forest Decision Tree