Explainable AI (XAI) with a Decision Tree

towardsdatascience.com/explainable-ai-xai-with-a-decision-tree-960d60b240bd

23 October 2021

Idit Cohen

Apr 18, 2021

A practical guide for XAI analysis with Decision Tree visualization

Image by Sager from

AI, Learning models become more and more complex over time and it becomes difficult to analyze them intuitively. It is often heard that machine learning models are “black boxes”, in the sense that they can make good predictions but we can’t understand the logic behind those predictions. This statement is true in the sense that most data scientists find it difficult to extract insights from models. However, there are a few tools that we can use to extract insights from sophisticated machine learning models.

This guide is a practical instruction on how to use and interpret the sklearn.tree.plot_tree for models explainability. A decision tree is an explainable machine learning algorithm all by itself and is used widely for feature importance of linear and non-linear models (explained in part global explanations part of this post). It is a relatively simple model, and it is easily explained by visualizing the tree.

```
import numpy as npfrom sklearn import treefrom sklearn.tree import
DecisionTreeClassifierimport matplotlib.pylab as pltfrom sklearn import datasets,
ensemble, model_selectionfrom sklearn.ensemble import RandomForestClassifier
```

In this example, we will use the breast cancer example from sklearn datasets. It is a simple binary (Malignant, Benign) classification problem[1]. Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe the characteristics of the cell nuclei present in the image.

```
# import data and splitcancer = datasets.load_breast_cancer()X_train, X_test,
y_train, y_test = model_selection.train_test_split(cancer.data, cancer.target,
random_state=0)
```

After splitting the dataset to train and test, use tree.DecisionTreeClassifier() to build the classification model.

```
# model and fitcls_t = tree.DecisionTreeClassifier()cls_t.fit(X_train, y_train);
```

Now, just to get a basic impression of the model, I recommend visualizing the feature importance. Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node[2]. 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. The most important features will be higher in the tree. A single feature can be used in different branches of the tree, feature importance then is its total contribution in reducing the impurity[3].

```
importances = cls_t.feature_importances_indices = np.argsort(importances)features
= cancer.feature_namesplt.title('Feature Importances')j = 11# top j
importanceplt.barh(range(j), importances[indices][len(indices)-j:], color='g',
align='center')plt.yticks(range(j), [features[i] for i in indices[len(indices)-
j:]])plt.xlabel('Relative Importance')plt.show()
```

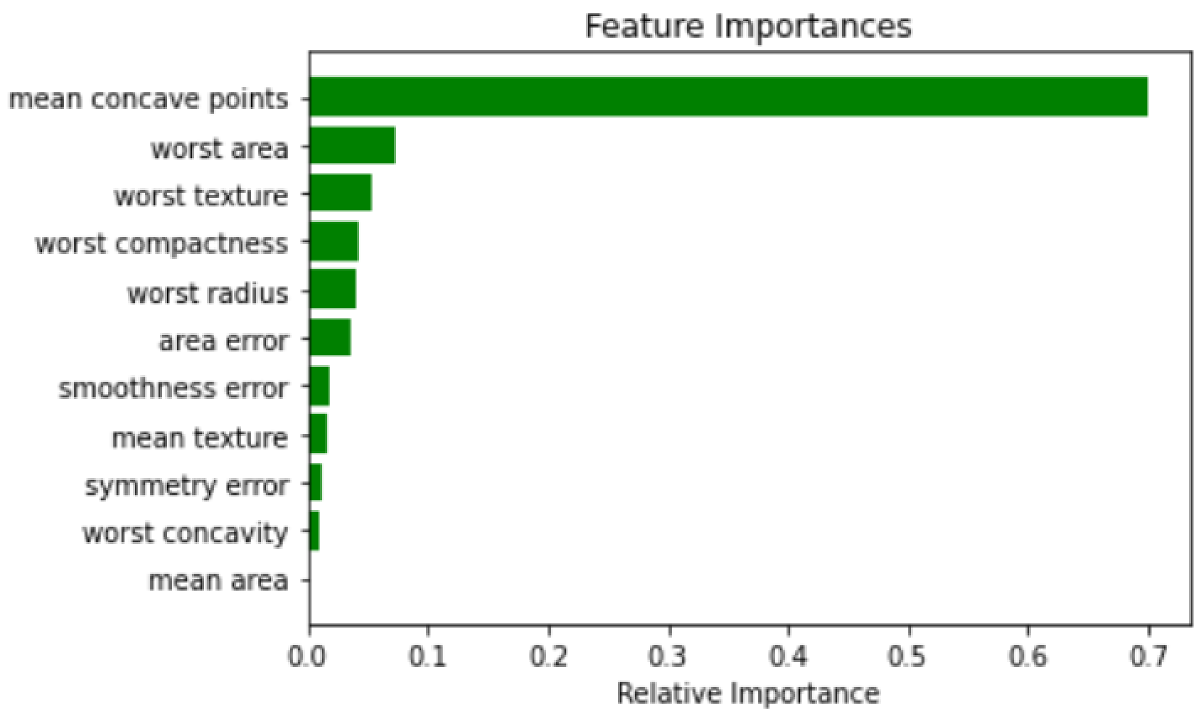


Image by Author

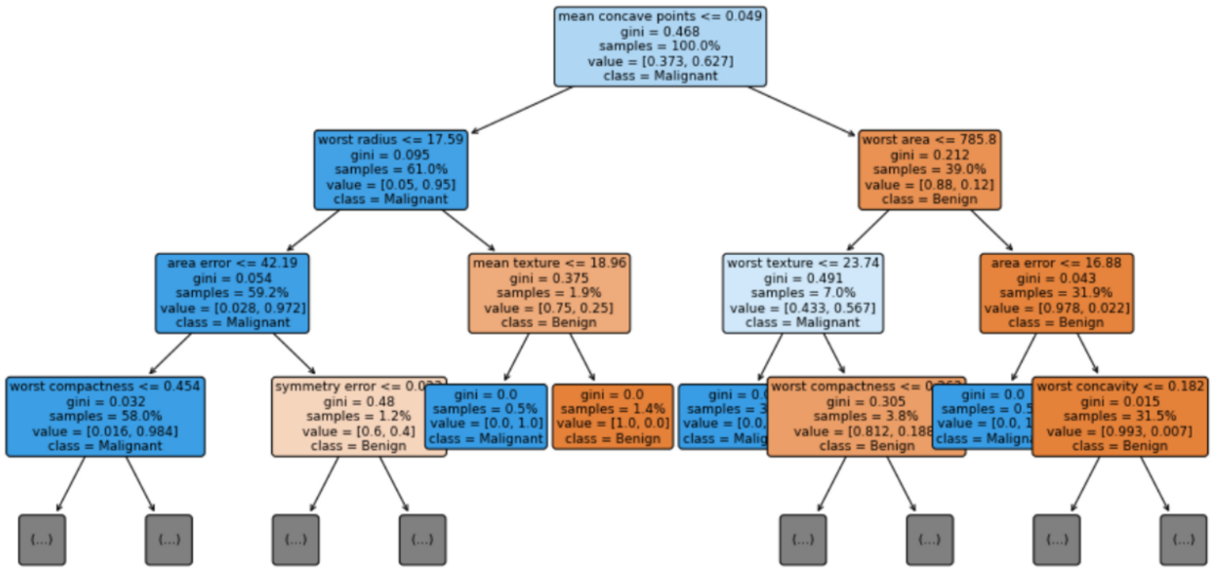
```
cls_t.feature_importances_
```

In this case, only the top 13 features are being used.The other features are not being used. Their importance is zero.

```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.07432987, 0.01832665, 0.        ,
       0.00145998, 0.        , 0.01330604, 0.01393509, 0.        ,
       0.00562787, 0.        , 0.        , 0.        , 0.00738236,
       0.        , 0.03657366, 0.70890614, 0.03397417, 0.00555788,
       0.        , 0.02482951, 0.05579079, 0.        , 0.        ])
```

Let us visualize the first three levels of the decision tree, max_depth=3.

```
# visualizationfig = plt.figure(figsize=(16, 8))vis = tree.plot_tree(cls_t,
feature_names = cancer.feature_names, class_names = ['Benign', 'Malignant'],
max_depth=3, fontsize=9, proportion=True, filled=True, rounded=True)
```



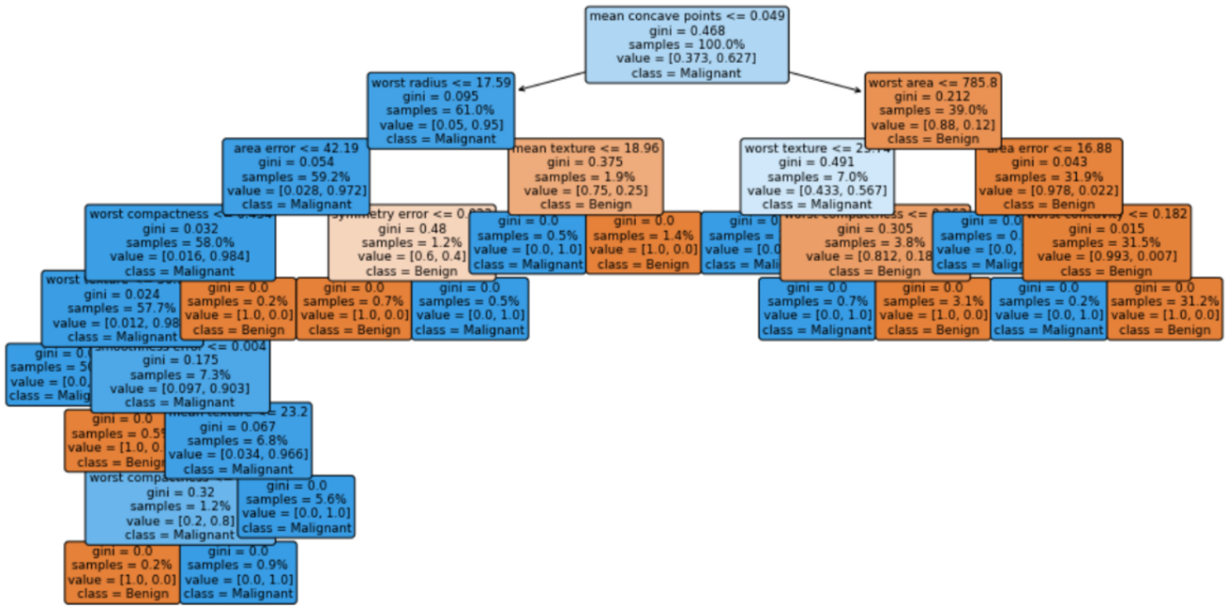
Decision tree visualization with max_depth=3 Image by Author

What can we learn about the model?

First, we can see for each decision level the name of the feature being used and the splitting value for the condition. If a sample satisfies the condition, then it goes to the left branch, otherwise, it goes to the right.

The samples line in each node shows us the number of samples we are examining in the current node. If proportion=True, the number in the samples line are in units, % of the total dataset.

The value line in each node is telling us how many samples at that node fall into each class, in order when proportion=False and the proportion of samples when proportion=True. That's why, in each node, the numbers in value add up to the number shown in value for proportion=False and 1 for proportion=True.



Decision tree visualization with max_depth=8 Image by Author

In the class line, we can see the classification result of the node.

The gini score is a metric that quantifies the purity of the node, similarly to entropy. A gini score greater than zero implies that samples contained within that node belong to different classes. In the above figure, the leaves have a gini score of zero, meaning that the samples in each leaf belong to a single class. Note that when purity is high the node/leaf has a darker color.

Decision Tree surrogate model

One popular way to explain the global behavior of a “black box” model is to apply the global surrogate model. A global surrogate model is an interpretable model that is trained to approximate the predictions of a black-box model. We can draw conclusions about the black-box model by interpreting the surrogate model. Solving machine learning interpretability by using more machine learning![4]

Training a surrogate model is a model-agnostic method since it does not require any information about the inner workings of the black-box model, only access to data and the prediction function is necessary. The idea is that we take our “black box” model and create predictions using it. Then we train a transparent model on the predictions produced by the “black box” model and the original features. Note that we need to keep track of how well the surrogate model approximates the “black-box” model but that is often not straightforward to determine.

Random Forest Classifier is a commonly used model that solves the overfit problem Decision Tree models tend to have. As a result, has better accuracy on the test set but it is

```
clf = RandomForestClassifier(random_state=42, n_estimators=50,
n_jobs=-1)clf.fit(X_train, y_train);
```

Create predictions using your model (in this case a RandomForestClassifier)

```
predictions = clf.predict(X_train)
```

Then use the predictions to fit the data to the Decision Tree Classifier.

```
cls_t = tree.DecisionTreeClassifier()cls_t.fit(X_train, predictions);
```

Visualize.

```
# visualizationfig = plt.figure(figsize=(16, 8))vis = tree.plot_tree(cls_t,
feature_names = cancer.feature_names, class_names = ['Benign', 'Malignant'],
max_depth=3, fontsize=9, proportion=True, filled=True, rounded=True)
```

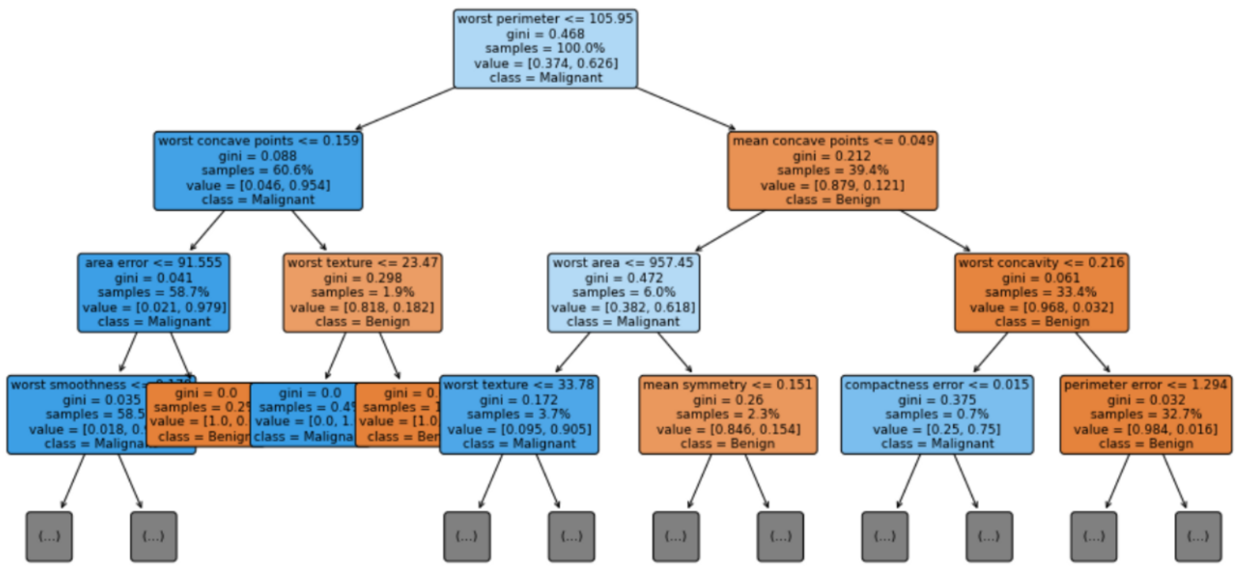


Image by Author

That's it! Even if we cannot easily comprehend how the hundreds of trees in the forest look, we can build a shallow decision tree instead and hopefully get an idea of how the forest works[5].

Lastly, Measure how well the surrogate model replicates the predictions of the black-box model. One way to measure how well the surrogate replicates the black-box model is the R-squared measure.

```
cls_t.score(X_train, predictions)
```

Tip

If you are using pycharm to create the model, you can export it to jupyter notebook by using pickle. On the export side:

```
import pickle# dump information to that filewith open('model','wb') as outfile:
pickle.dump(cls_t, outfile)
```

And in the import side:

```
import pickle# load information from that filewith open('model','rb') as
inputfile:    modell = pickle.load(inputfile)
```

Summary

Explaining "black box" machine learning models is important for their successful applicability to many real-world problems. sklearn.tree.plot_tree is a visualization tool that can help us understand the model. Or in other words, what did the machine (model) learn from the features? Does it match our expectations? Can we help the machine learn by adding more complex features using domain knowledge about the problem? Using decision tree visualization can help us assess the correctness of the model intuitively and perhaps even improve it. The code in this post can be found here.