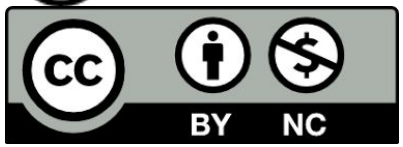
Learn ai

Decision Trees

Lesson 6: Scikit learn

An AI Commons initiative



What are decision trees?

- Decision tree algorithms provide a predictive model for classification or regression
  - Classification uses information gain
  - Regression uses standard deviation reduction
- They build a top down hierarchical classification by splitting data along criteria in a tree structure
  - Criteria nodes carry the decision making
  - Leaf nodes carry the values
- They constitute the basis for random forest ensemble learning

Decision trees pros

- They are convenient
  - They are simple to understand and to interpret. Trees can be visualised.
  - They mimic the decision making in business, and decision can be explained.
  - They are able to handle multi-output problems.
  - They can handle both numerical and categorical data.
    - However scikit-learn implementation does not support categorical variables for now.
- They are cheap
  - They require little data preparation.
    - Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed.
  - The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
  - They tend to perform well even if its assumptions are somewhat violated by the true model from which the data were generated.
-

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

Decision trees cons

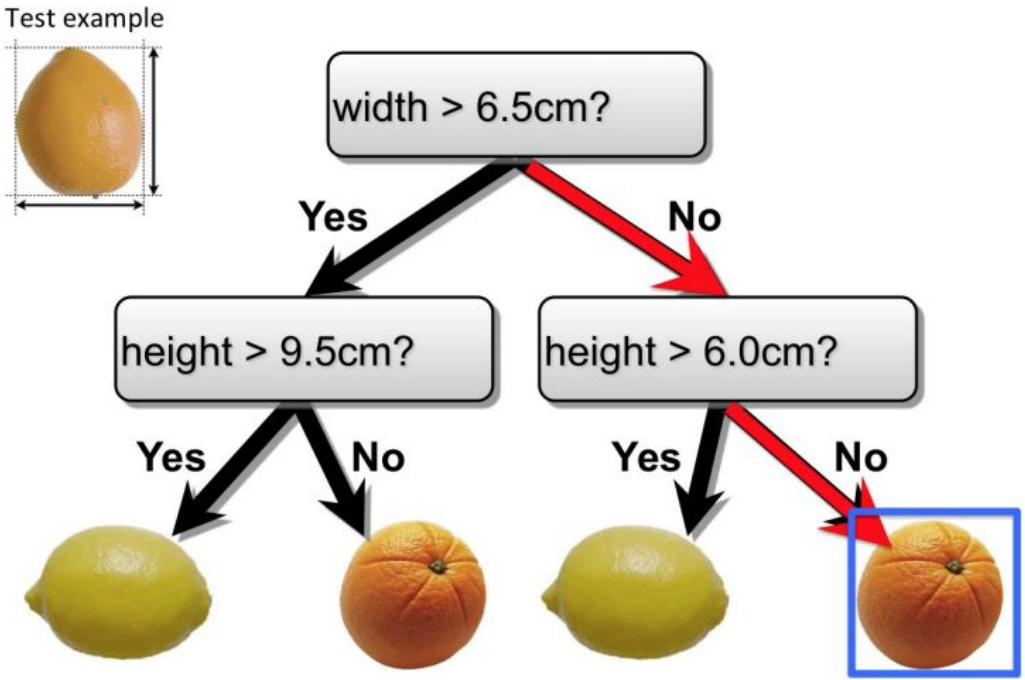
- They are not the most accurate
  - Compared to other supervised learning techniques
- They can get unstable
  - Small change in the data can lead to big change in the structure of the tree.
- Deep trees have a tendency to overfit.
  - Random-forest ensemble methods are used to minimize overfit.
- They can create bias
  - If the dataset classes are not balanced
- They are not great for continuous variables
  - They provide piecewise approximation
- Some patterns are hard to learn
  - Switching patterns like multiplexing, or exclusive OR logic

Let's look at an example

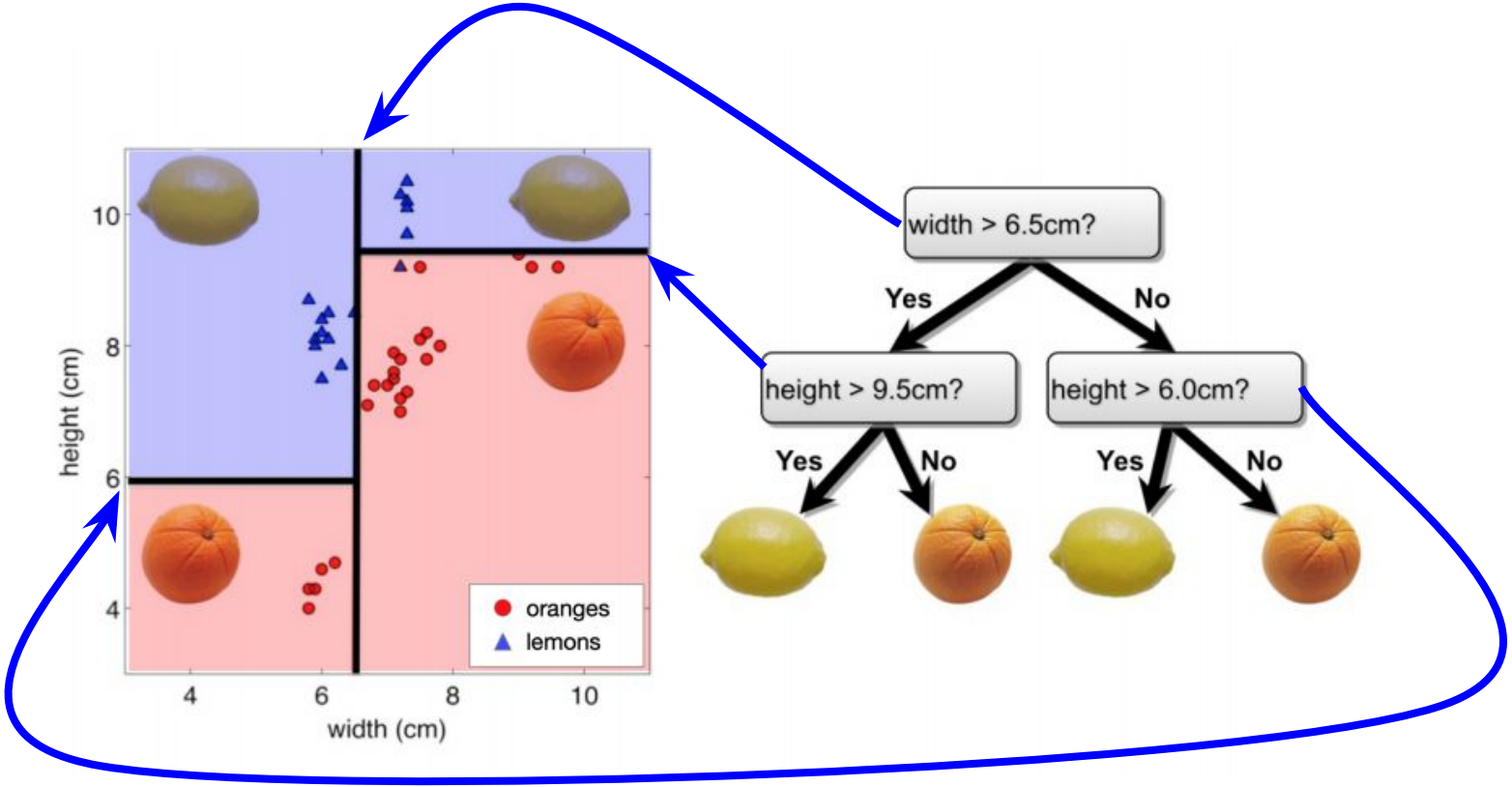
Dataset: number of lemons and oranges

Features: Width & Height

We are splitting on width first
Then we are splitting on Height

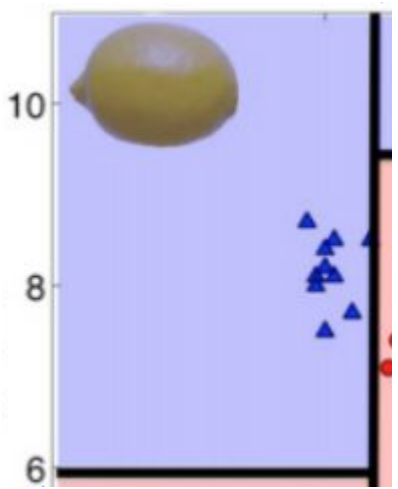


Let's look at an example



Majority voting

- Here, we have the trained decision tree.
- We have created the partitions
- But how do we know what each partition is labeled as what? how do we know any data point in this partition will be a lemon?
- We take a majority vote

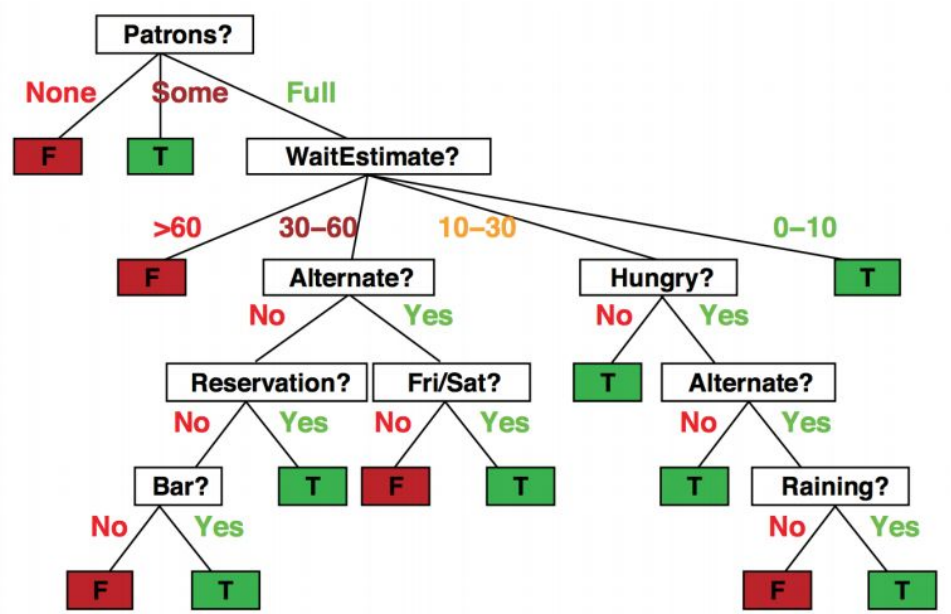


Let's look at an example 2.0

Learn ai

Type: *Df +D*

- Will I eat at this restaurant?



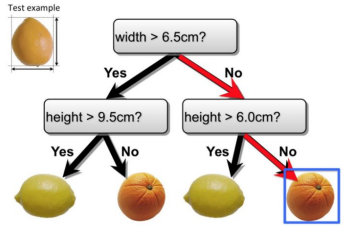
AI

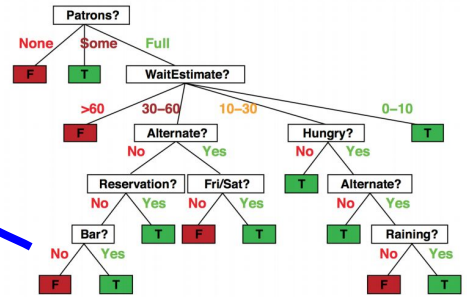
Classification v/s Regression

Regression
DecisionTreeRegressor

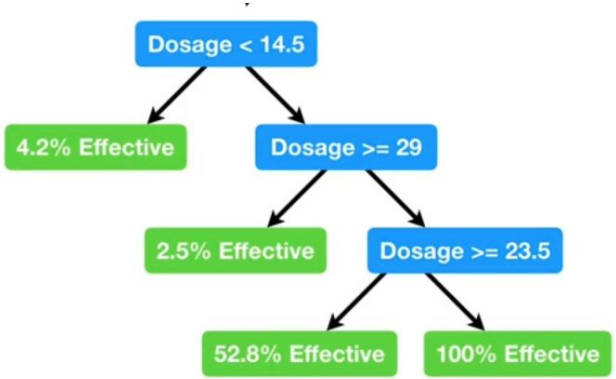
Classification
DecisionTreeClassifer

	Real Valued Outputs	Discrete Outputs
Real Valued Features	Rf + R	Rf + D
Discrete Features	Df + R	Df + D

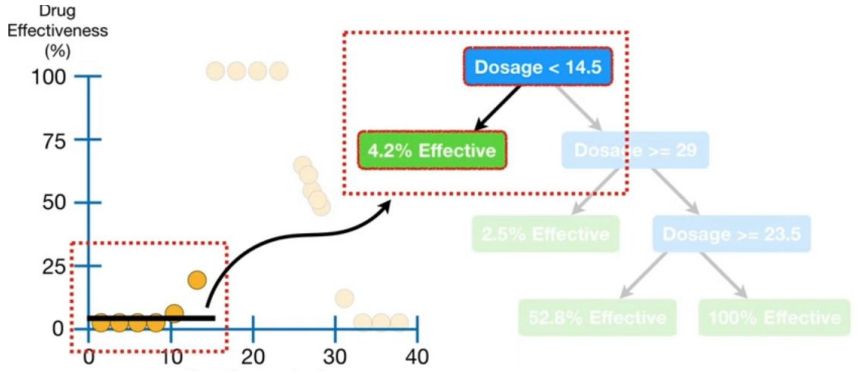




Decision Regression Trees (for real-valued outputs)



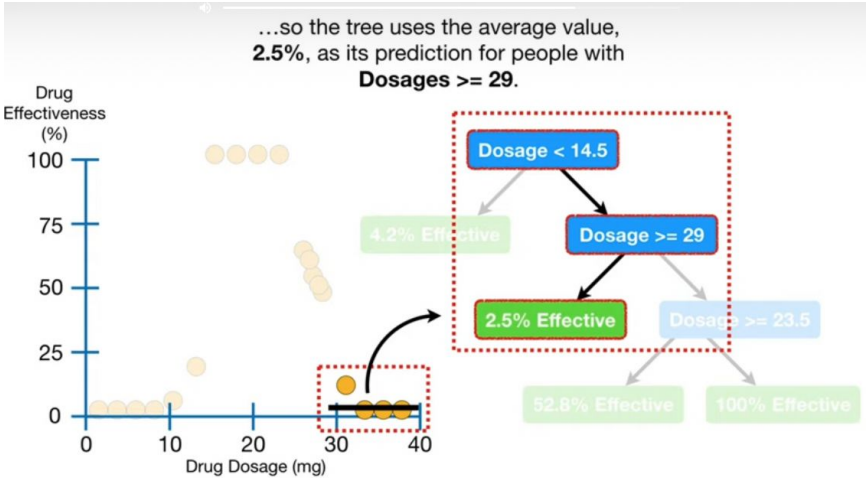
⇒



Type: *Rf + R*

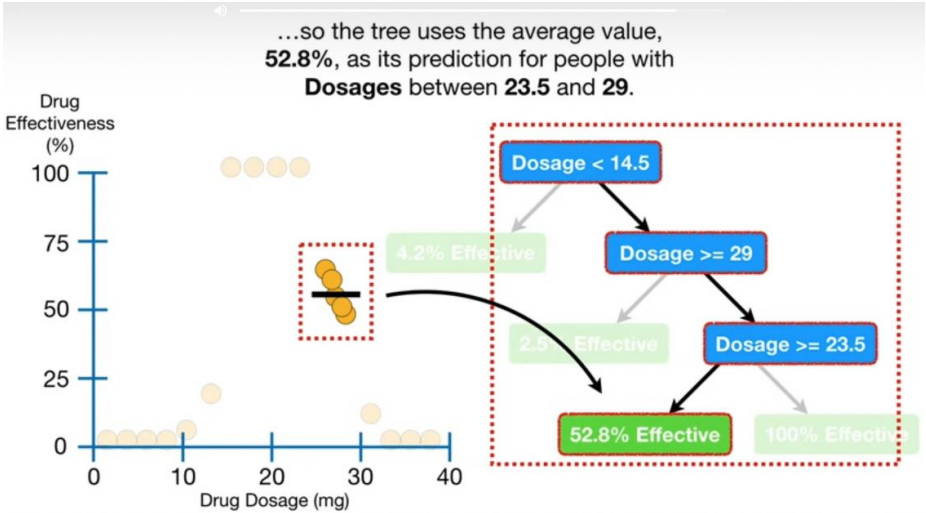
Dosage < 14.5 ⇒ take the mean of all points where dosage is less than 14.5
This is the prediction. If Dosage < 4.5, the regression tree prediction is 4.2%

Decision Regression Trees (for real-valued outputs)



Dosage ≥ 14.5 ⇒ take Dosage < 29. Take the mean of all points where dosage is greater than 14.5 but less than 29
This is the prediction. If Dosage ≥ 14.5 & Dosage < 29,, the regression tree prediction is 2.5%

Decision Regression Trees (for real-valued outputs)



Dosage ≥ 14.5 ⇒ take Dosage < 29 ⇒ Dosage ≥ 23.5 . Take the mean of all points where dosage is greater than 23.5 but less than 29.
This is the prediction. If Dosage ≥ 23.5 & Dosage < 29,, the regression tree prediction is 52.8%

What features to use?

- So we are at the root node. What feature do we split on?
- How do we know what the best feature to start with is?
- We use a measure called the Gini Index and/or Entropy

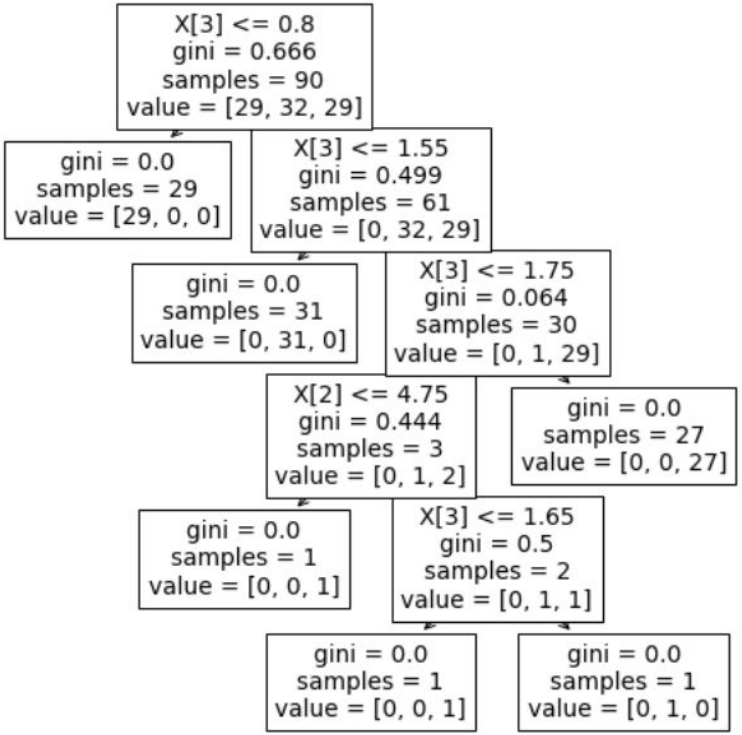
What is the Gini Index/Entropy?

- Gini index/Entropy is the information gain of a feature
- At the root node, we take the feature that gives us the highest information gain
- When we have continuous features(Ex: Weight & Height), the need the best thresholds of the best feature as well
- Gini/Entropy chooses the feature and threshold that give the highest information gain.
- When we have discrete features, just the feature that gives the highest information gain is enough

Gini vs Entropy

- Both measure information gain of a feature
- Gini is faster
- Entropy is slightly better quality

Decisition Trees w/ Gini



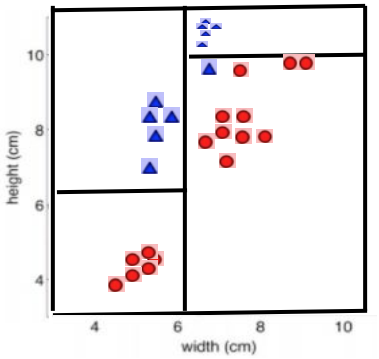
Learning Decision Trees

1. Start with the whole training set and an empty decision tree
2. Choose the feature that provides the highest information gain
3. Split on the feature and recurse on subpartitions

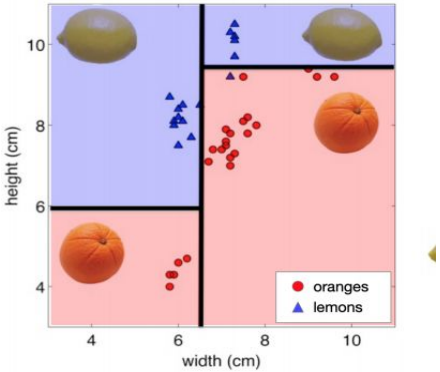
Summary

Learn DT

Step 1) Start with the whole training set and an empty decision tree
Step 2) Choose the feature that provides the highest information gain
Step 3) Split on the feature and recurse on subpartitions



Now, we have a learned DT with the partitions.



Take the majority vote in each partition, and label the partition accordingly

Summary (Important things to note)

1. Decision Trees are for Classification & Decision Regressors are for Regression
2. Gini/Entropy only applies to Decision Trees, and not Decision Tree Regressor
3. Once we have created the partitions:
    a. Discrete Outputs: Majority vote
    b. Continuous Outputs: Mean of all data points in that partition
4. Trees with Continuous Outputs is Regression, whereas Trees with Discrete Outputs is Classification
5. With Real-valued features, Gini/Entropy not only gets the best feature (highest information gain), but also the best threshold for those features

Tips On Practical Use From Sklearn

- 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.
- Visualise 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.
- Remember that the number of samples required to populate the tree doubles for each additional level the tree grows to. Use `max_depth` to control the size of the tree to prevent overfitting.
- 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. If the sample size varies greatly, a float number can be used as percentage in these two parameters. While `min_samples_split` can create arbitrarily small leaves, `min_samples_leaf` guarantees that each leaf has a minimum size, avoiding low-variance, over-fit leaf nodes in regression problems. For classification with few classes, `min_samples_leaf=1` is often the best choice.

  Note that `min_samples_split` considers samples directly and independent of `sample_weight`, if provided (e.g. a node with m weighted samples is still treated as having exactly m samples). Consider `min_weight_fraction_leaf` or `min_impurity_decrease` if accounting for sample weights is required at splits.

Tips On Practical Use From Sklearn

- Balance your dataset before training to prevent the tree from being biased toward the classes that are dominant. Class balancing can be done by sampling an equal number of samples from each class, or preferably by normalizing the sum of the sample weights (`sample_weight`) for each class to the same value. Also note that weight-based pre-pruning criteria, such as `min_weight_fraction_leaf`, will then be less biased toward dominant classes than criteria that are not aware of the sample weights, like `min_samples_leaf`.
- If the samples are weighted, it will be easier to optimize the tree structure using weight-based pre-pruning criterion such as `min_weight_fraction_leaf`, which ensure that leaf nodes contain at least a fraction of the overall sum of the sample weights.
- All decision trees use `np.float32` arrays internally. If training data is not in this format, a copy of the dataset will be made.
- If the input matrix X is very sparse, it is recommended to convert to sparse `csc_matrix` before calling fit and sparse `csr_matrix` before calling predict. Training time can be orders of magnitude faster for a sparse matrix input compared to a dense matrix when features have zero values in most of the samples.

Python Implementation

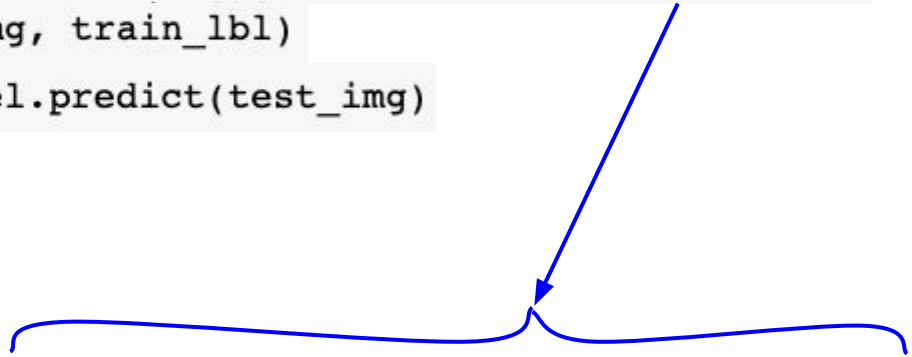
Learn ai

Instantiate the model → `mnist_tree_model = tree.DecisionTreeClassifier(criterion="entropy")`

Fit the model → `mnist_tree_model.fit(train_img, train_lbl)`

Make predictions → `predictions = mnist_tree_model.predict(test_img)`

1) Instantiates an empty decision tree with the information gain criterion specified
2) We fit the model using the training set and it's labels. In this step, the model builds the decision tree selecting features using the criterion specified
3) We make predictions. We take the trained decision tree to classify each data point in the testing dataset



criterion : *{"gini", "entropy"}, default="gini"*

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

AI

Python Implementation

apply(X[, check_input])	Return the index of the leaf that each sample is predicted as.
cost_complexity_pruning_path(X, y[, ...])	Compute the pruning path during Minimal Cost-Complexity Pruning.
decision_path(X[, check_input])	Return the decision path in the tree.
fit(X, y[, sample_weight, check_input, ...])	Build a decision tree classifier from the training set (X, y).
get_depth()	Return the depth of the decision tree.
get_n_leaves()	Return the number of leaves of the decision tree.
get_params([deep])	Get parameters for this estimator.
predict(X[, check_input])	Predict class or regression value for X.
predict_log_proba(X)	Predict class log-probabilities of the input samples X.
predict_proba(X[, check_input])	Predict class probabilities of the input samples X.
score(X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
set_params(**params)	Set the parameters of this estimator.

DecisionTreeClassifier

apply(X[, check_input])	Return the index of the leaf that each sample is predicted as.
cost_complexity_pruning_path(X, y[, ...])	Compute the pruning path during Minimal Cost-Complexity Pruning.
decision_path(X[, check_input])	Return the decision path in the tree.
fit(X, y[, sample_weight, check_input, ...])	Build a decision tree regressor from the training set (X, y).
get_depth()	Return the depth of the decision tree.
get_n_leaves()	Return the number of leaves of the decision tree.
get_params([deep])	Get parameters for this estimator.
predict(X[, check_input])	Predict class or regression value for X.
score(X, y[, sample_weight])	Return the coefficient of determination R^2 of the prediction.
set_params(**params)	Set the parameters of this estimator.

DecisionTreeRegressor

References

- https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier
- https://scikit-learn.org/stable/modules/tree.html#tree
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html#sklearn.tree.DecisionTreeRegressor

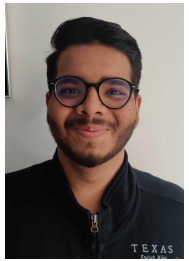
Contributors

Mustafa Imam

Addison Weatherhead

Isha Sharma

Kevin Zhu



Pranjal Kumar

Gilles Fayad