**Big Data Analysis**

**Predicting Forest Cover with Decision Trees**

Ji-Hyeong Han

( [jhhan@seoultech.ac.kr](mailto:jhhan@seoultech.ac.kr) )

Dept. of Computer Science and Engineering



# Decision Tree

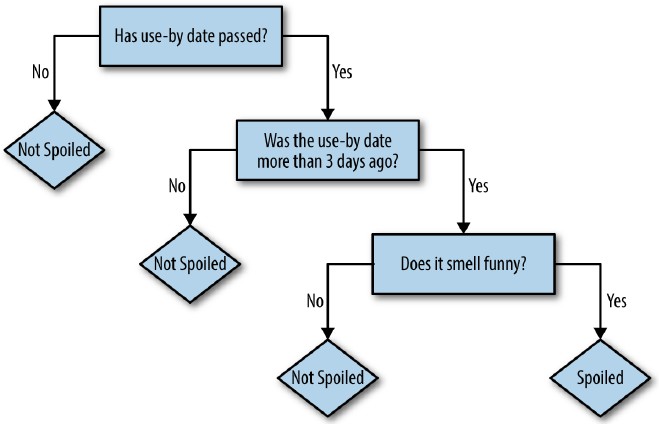
2

* Is the milk spoiled?

Node

**Introduction to Decision Tree**

Leaf



Edge

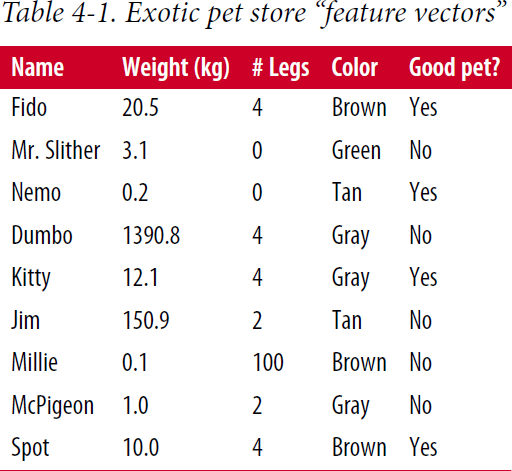
3

**Introduction to Decision Tree**

* A robot has taken a job in an exotic pet store.

It wants to learn which animals in the shop would make a good pet for a child.

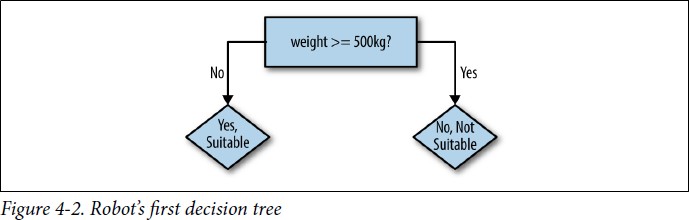
The owner lists nine pets that would and wouldn’t suitable

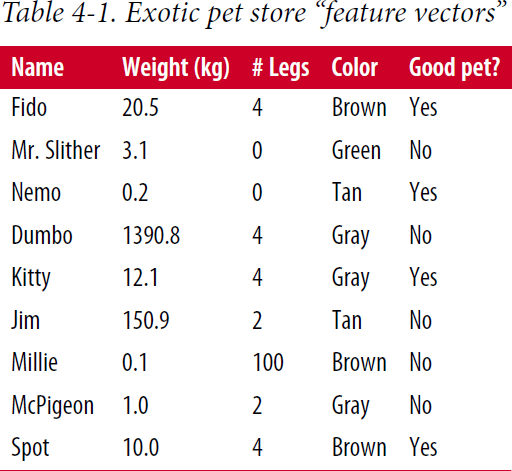


4

**Introduction to Decision Tree**

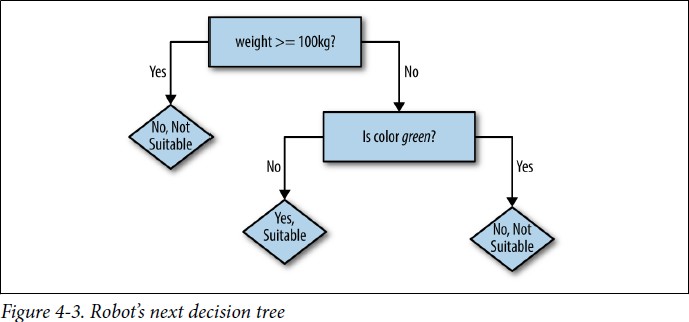
* The robot might try to fit a simple decision tree to this training data

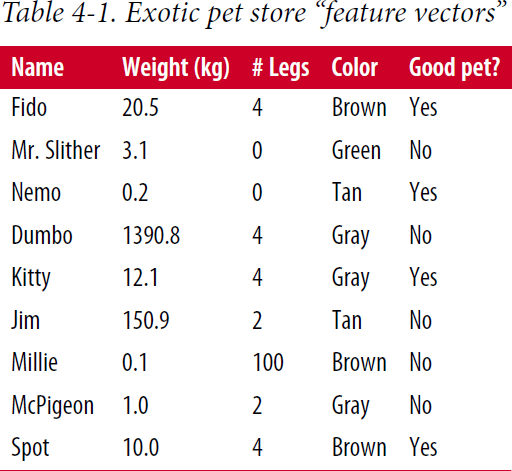


* + The heavy animals are predicted correctly
  + The lighter animals are only partly correct

5

**Introduction to Decision Tree**



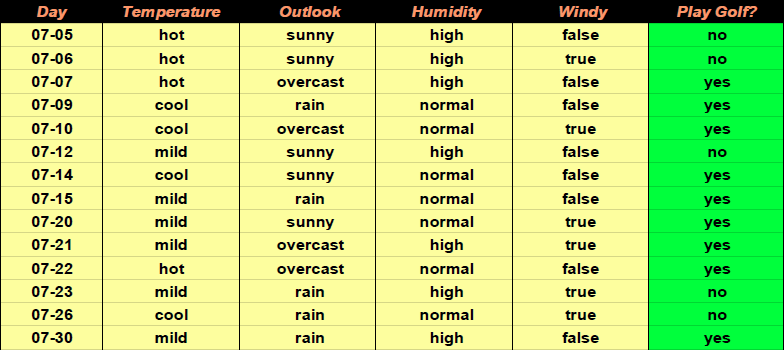
* Now, seven of nine examples are correct
* Of course, decision rules could be added until all nine were correctly predicted
* But, we need a balance to prevent the

*overfitting* to the training data

6

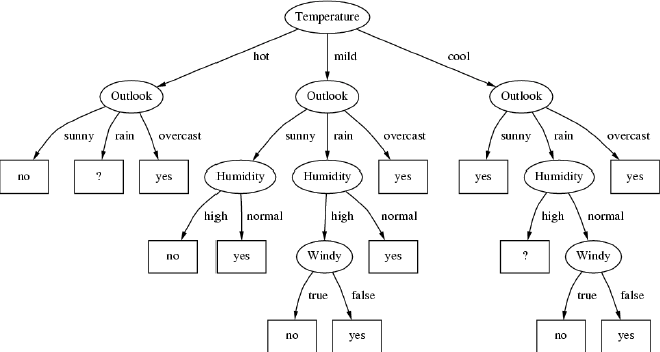
**Decision Tree Learning Algorithm**

* Sample task



7

**Decision Tree Learning Algorithm**

* Decision tree example
* It can explain all of the training data
* Will it generalize well to new data?

8

**Decision Tree Learning Algorithm**

* We want to grow a simple tree

- We prefer attributes that split the data so that each successor node is as pure as possible

* So, we prefer that each node mostly contains examples of a single class
* We want to a measure this preference
* Good: All examples are of the same class
* Bad: All classes are equally likely

→ Entropy is a proper measure for this

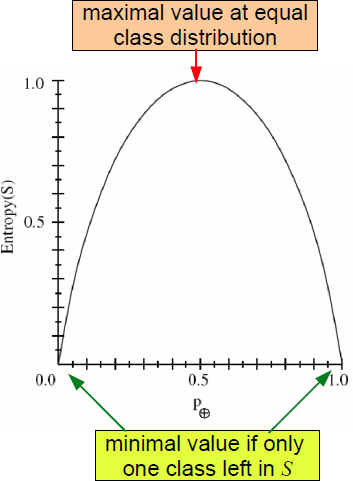
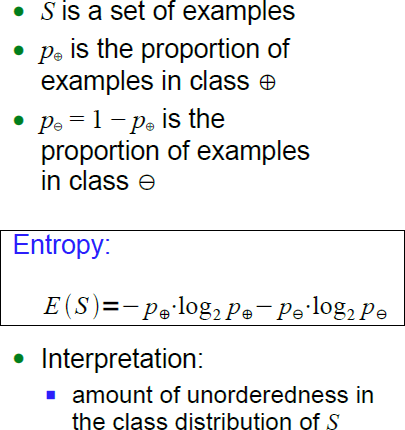
* Entropy is 0, when all examples are of the same class



9

**Decision Tree Learning Algorithm**

* Entropy for two classes

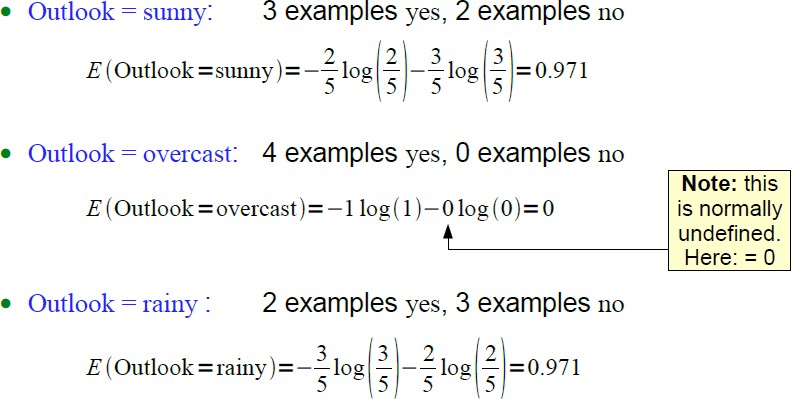


10

**Decision Tree Learning Algorithm**

* Entropy example

2 3



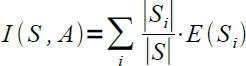
3 2

11

**Decision Tree Learning Algorithm**

* Entropy only computes the quality of a single set of examples
* How can we compute the quality of the entire split?

→ Compute the weighted average over all sets resulting from the split



* Example

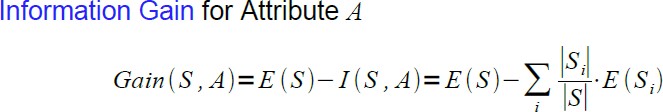
- Average entropy for attribute *Outlook*



12

**Decision Tree Learning Algorithm**

* Information Gain
* When an attribute *A* splits the set *S* into subsets *Si*
* We compute the average entropy
* And compare the sum to the entropy of the original set *S*



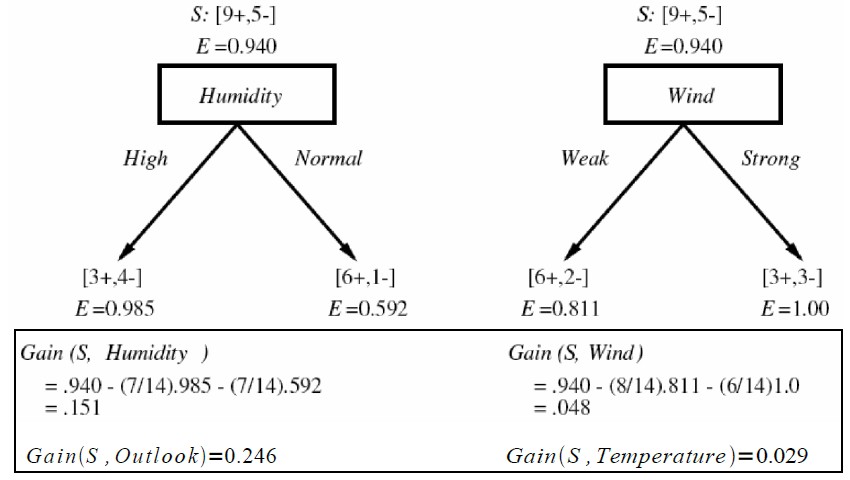
* The attribute that maximizes the difference (information gain) is selected
* Maximizing information gain is equivalent to minimizing average entropy

13

**Decision Tree Learning Algorithm**

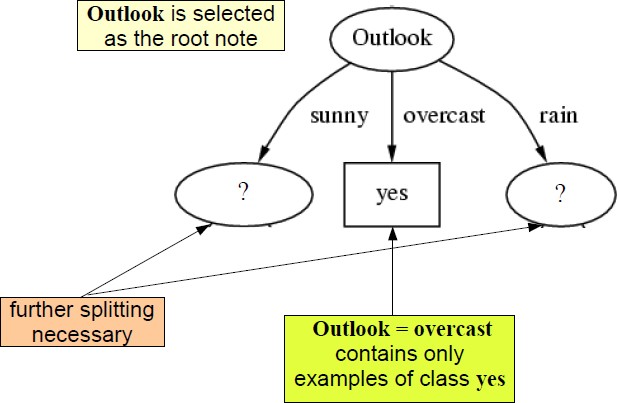
* Example

14



**Decision Tree Learning Algorithm**

* Example (cont’d)



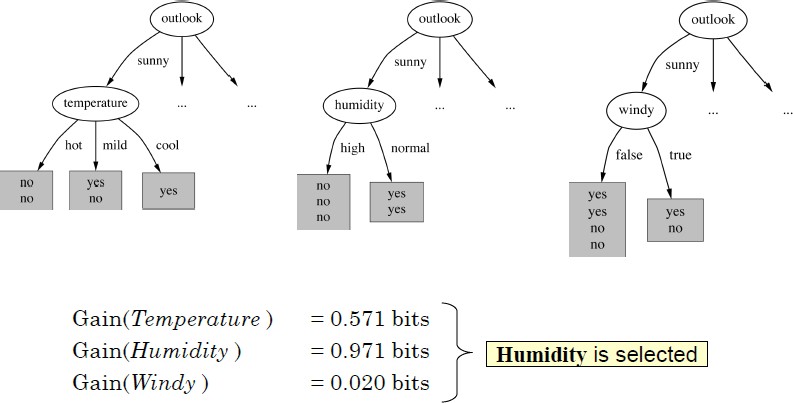
node

15

**Decision Tree Learning Algorithm**

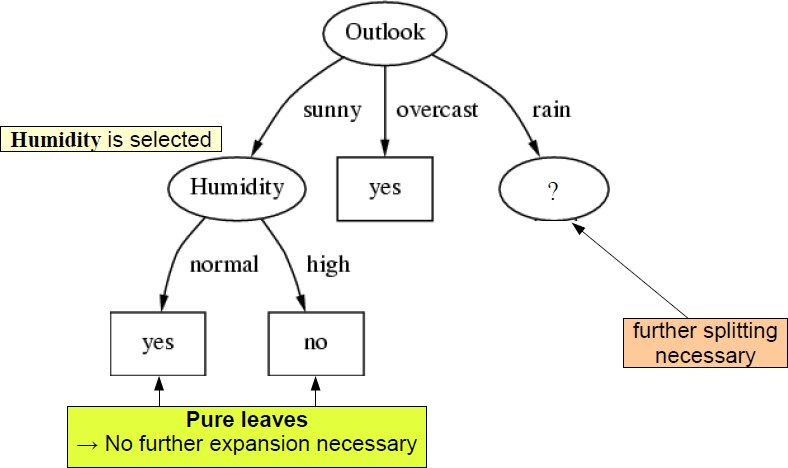
* Example (cont’d)

yes no no

16

**Decision Tree Learning Algorithm**

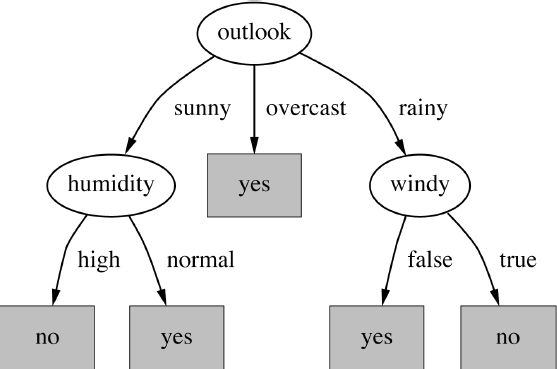
* Example (cont’d)



17

**Decision Tree Learning Algorithm**

* Example – final decision tree



18

**Introduction to Random Forest**

* Random Forest (or Random Decision Forest)

- Ensemble model by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes of the individual trees

* The prediction of a random forest is simply a weighted average of the trees’ predictions

- For a categorical target (i.e. classification), this can be a majority vote or the most probable value based on the average of probabilities produced by the trees

* Random forests are appealing in the context of big data because trees are supposed to be built independently, and big data technologies like Spark and Map-Reduce inherently need data- parallel problems.

19

# Practice

20

**Tutorial of decision tree & random forest**

* [https://spark.apache.org/docs/2.0.0-preview/ml-classification- regression.html#decision-tree-classifier](https://spark.apache.org/docs/2.0.0-preview/ml-classification-regression.html#decision-tree-classifier)
* “data/mllib/sample\_libsvm\_data.txt” is in github [https://github.com/apache/spark/blob/master/data/mllib/sample\_libsv m\_data.txt](https://github.com/apache/spark/blob/master/data/mllib/sample_libsvm_data.txt)

21

**Practice**

* Download the data

<https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/>

* unzip “covtype.data.gz”
* using “covtype.data” file

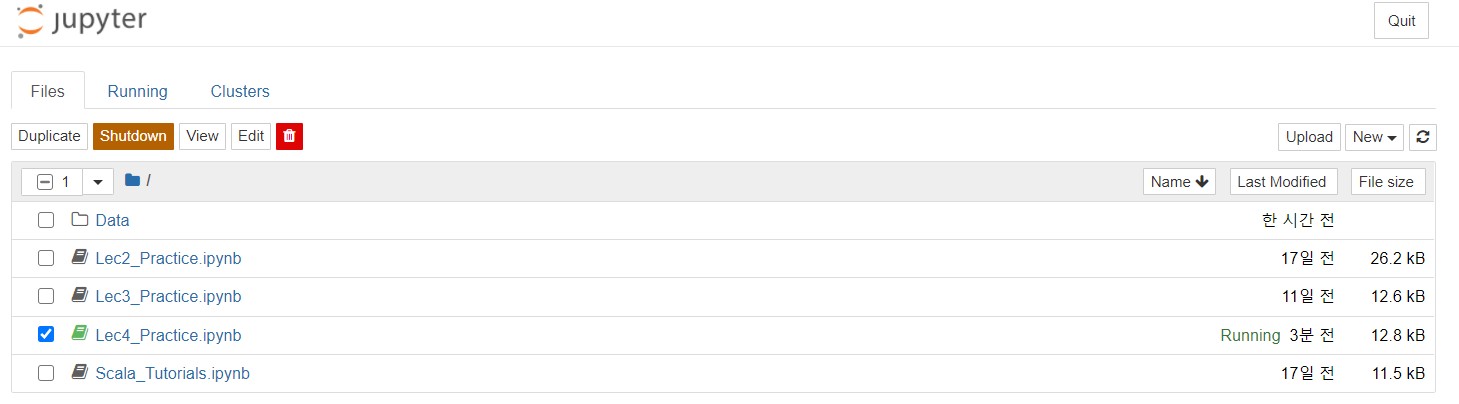
- upload “covtype.data” to the jupyter notebook

* “covtype.info” gives information about the data
* The data set is about the types of forest-covering in Colorado, USA
* It’s only a coincidence that data set concerns real-world forest!!
* Each data example contains 54 features (for example, slope, distance to water, shade, soil type)
* There 7 different labels (classes) of tree type
  + It is our target class which needs to be classified

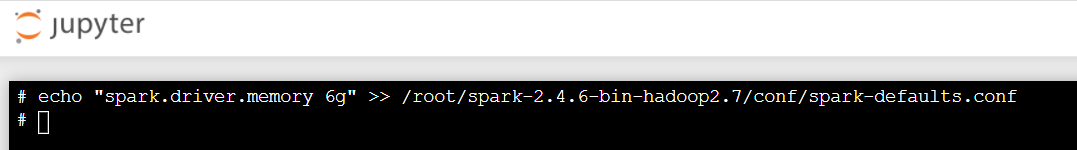
22

**Practice**

* Add memory to spark docker image
* First, shut down any running notebook files



* Type commands in terminal

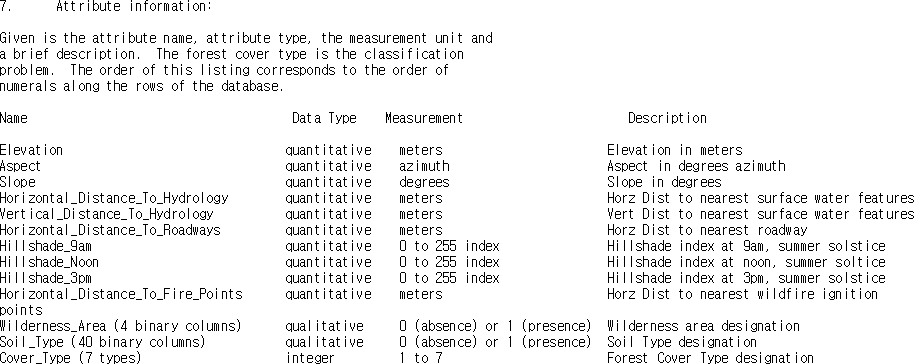


* If you want to add more cores, then you can do the similar things # echo “spark.driver.cores 8” >> /root/spark-2.4.6-bin-

hadoop2.7/conf/spark-defaults.conf 23

**Practice**

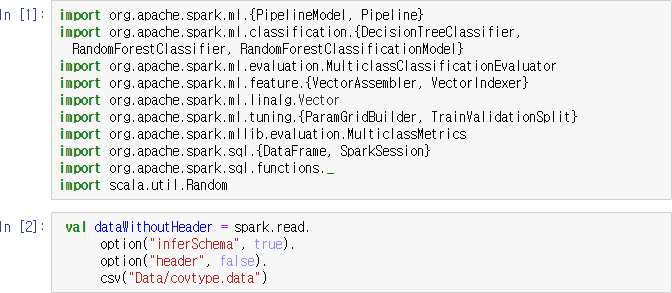
* Data description from info file

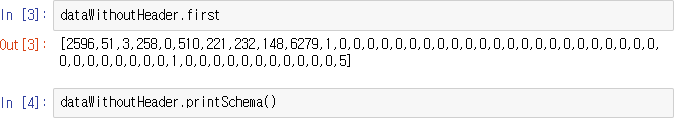


24

**Practice**

* Import several libraries
* Read the data file
* Check the first data and schema

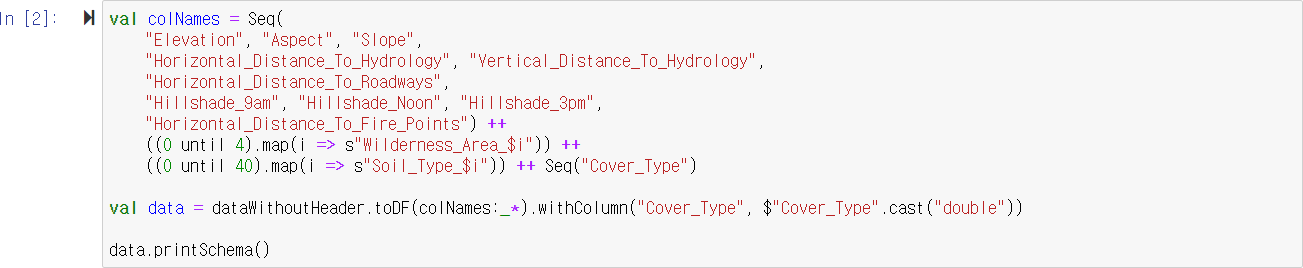


25

**Practice**

* Add column names and check the updated schema

- You should look at the info file to get the right column names



26

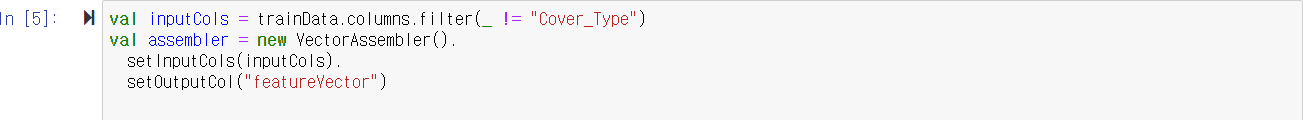
**Practice**

* Split the train and test data



* To use MLlib, we need to make all input feature columns into one column, whose value is a vector.

We can use *VectorAssembler Class*



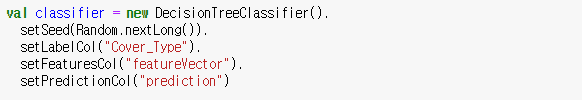
* We can make the new train data using assembled vector



27

**Practice**

* We make the decision tree model



* Train the model and see the trained decision tree structure



* Decision trees are able to access the importance of input features We can estimate how much each input feature contributes to making correct predictions

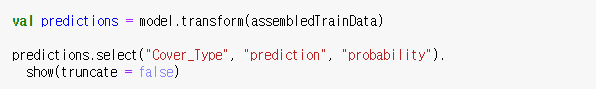


28

**Practice**

* We can see the prediction result from the training data (not only test data)

Because by doing this, we can recognize the trained model is fitted to the training data



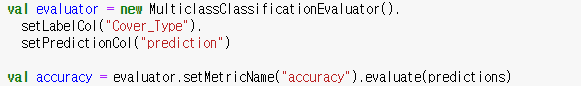
Maybe prediction result is not very good, because we don’t tune the hyperparameters

29

**Practice**

* Evaluate the trained model

Using *MulticlassClassifierEvaluator* to compute the accuracy and other metrics that evaluate the quality of the model’s predictions

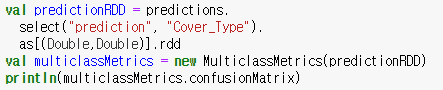


30

**Practice**

* Another good way to see the quality of model’s predictions is using confusion matrix
* Confusion matrix
* A table with a row and a column for every possible target values
* In our case, this is a 7 x 7 matrix
* Each row corresponds to an actual correct value Each column corresponds to a predicted value
* First, we need to convert the DataFrame to RDDs to calculate the confusion matrix

And then, we can calculate the confusion matrix



31

**Practice**

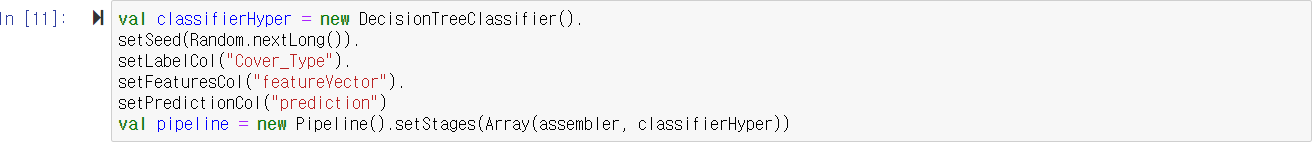
* Now, we are going to tune the hyperparameters of decision tree
* Hyperparameters of decision tree
* Maximum depth
  + Limits the number of levels in the decision tree
* Maximum bins
  + Bins mean the set of decision rules
* Impurity measure
  + Same concept with Entropy
* Minimum information gain
  + We already learned about it from the previous slides

32

**Practice**

* To tune the hyperparameters, we’re going to set up a pipeline encapsulating the two steps

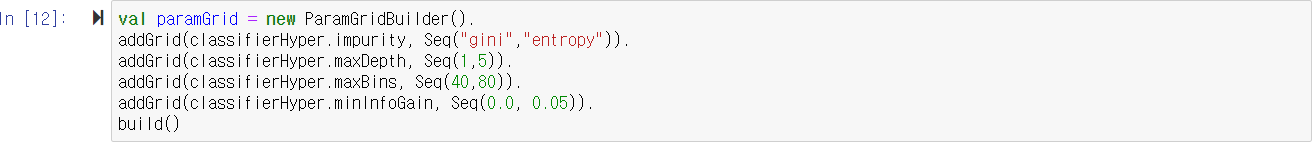
(*VectorAssembler* and *DecisionTreeClassifier*)



33

**Practice**

* Now, we’re going to define the combinations of hyperparameters by using *ParamGridBuilder*
* And also we need to define the evaluation metric that will be used to pick the “best” hyperparmeters

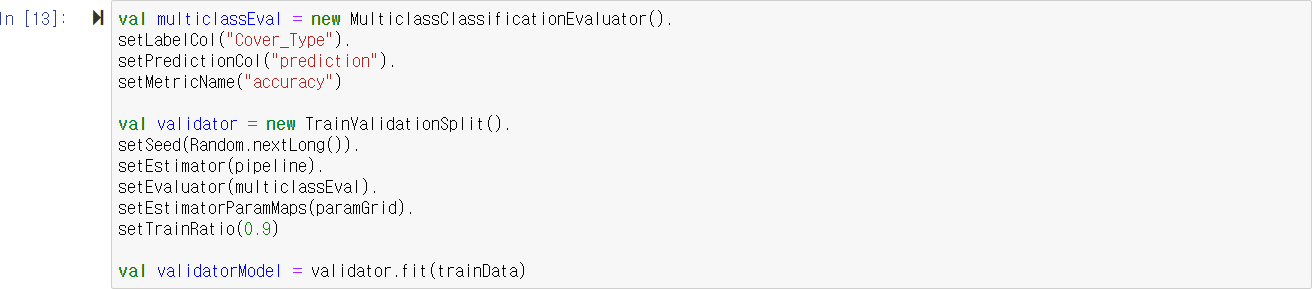


* This means that we’re going to evaluate 16 models (two values of four hyperparmeters)

34

**Practice**

* Finally, *TrainValidationSplit* brings all components together (i.e. the pipeline that makes models, model evaluation metrics, and hyperparmeters to try)
* TrainRatio means that we’re going to use 90% data to train each model and use 10% data to evaluate the model
* This will take long time, because it’s building and evaluating many models
* If kernel dies, please reduce the number of hyperparameters combinations, add more cores, or add more memories



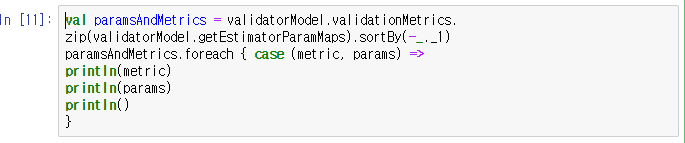
35

**Practice**

* The result of validator contains the best model it found



* To see the accuracy that each of the models achieved for each combination of hyperparmeters



36

**Practice**

* Finally, we’re going to use the model with the best hyperparmeters to predict the test data and see the accuracy result



- The result will be much better than the previous one which we don’t tune the hyperparameters

37