CS 7265: Big Data Analytics
Homework 3

You will build a decision tree from the Covetype dataset: Covtype.data and covtype.info from (https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/) and output a precision.

Input: covetype.data

Output: a precision value from your decision tree. **Optimize Impurity, maxBins and other hyper-parameters to achieve the highest precision values in the testing dataset.**

Ideas: You first load the CSV dataset, parse, and convert non-numeric data to numeric. The covtype.info file says that four of the columns are actually a one-hot encoding of a single categorical feature, called Wilderness_Type, with four values. Likewise, 40 of the columns are really one Soil_Type categorical feature. The target itself is a categorical value encoded as the values 1 to 7.

To start, the data will be used as is. The DecisionTree implementation, like several in Spark MLlib, requires input in the form of LabeledPoint objects:

```
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.regression._

val rawData = sc.textFile("hdfs:///user/ds/covtype.data")

val data = rawData.map { line =>
  val values = line.split(',').map(_.toDouble)
  val featureVector = Vectors.dense(values.init) ❶
  val label = values.last - 1 ❷
  LabeledPoint(label, featureVector)
}
```

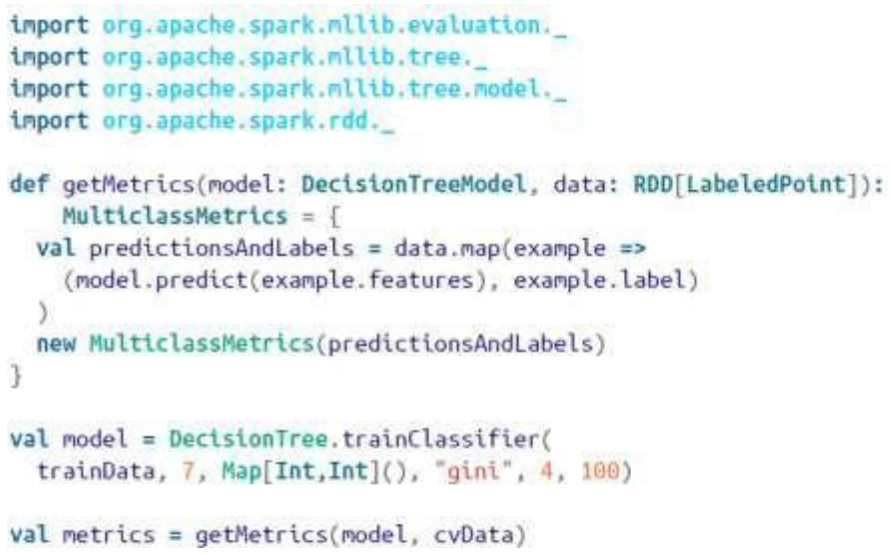
❶ init returns all but last value; target is last column

❷ DecisionTree needs labels starting at 0; subtract 1

Split the dataset: 80% of the data is used for training, and 10% each for cross-validation and test:

```
val Array(trainData, cvData, testData) =
  data.randomSplit(Array(0.8, 0.1, 0.1))
trainData.cache()
cvData.cache()
testData.cache()
```

A Decision Tree implementation has several hyper-parameters for which a value must be chosen. The training and CV sets are used to choose a good setting of these hyper-parameters for this data set. Here, the third set, the test set, is then used to produce an unbiased evaluation of the expected accuracy of a model built with those hyper-parameters. The accuracy of the model on just the cross-validation set tends to be biased and slightly too optimistic.



```
import org.apache.spark.mllib.evaluation._
import org.apache.spark.mllib.tree._
import org.apache.spark.mllib.tree.model._
import org.apache.spark.rdd._

def getMetrics(model: DecisionTreeModel, data: RDD[LabeledPoint]):
    MulticlassMetrics = {
  val predictionsAndLabels = data.map(example =>
    (model.predict(example.features), example.label)
  )
  new MulticlassMetrics(predictionsAndLabels)
}

val model = DecisionTree.trainClassifier(
  trainData, 7, Map[Int,Int](), "gini", 4, 100)

val metrics = getMetrics(model, cvData)
```

The number of target values it will encounter is 7. The Map holds information about categorical features; in this case, no features are treated as categorical features. The “gini” is the impurity measure. You may use “entropy” as well. The maximum depth is 4, and the maximum bin count is 100.

MulticlassMetrics computes standard metrics that in different ways measure the quality of the predictions from a classifier, which here has been run on the CV set. Ideally, the classifier should predict the correct target category for each example in the CV set. The metrics available here measure this sort of correctness, in different ways.

Precision is a metrics that shows the portion of correct predicted class in the total number of samples being predicted in that class. For example, if there are 100 samples in the CV set that are predicted in class 0 and 70 of them are actually in class 0, the precision is 70%. In this assignment, there are 7 classes and there are 7 precision values. You may use *metics.precision(i)* to find out the precision of class *i*. To find out overall precision, we will need to find the portion of samples in each class in the CV set and compute the weighted average using the following formula:

$$OverallPrecision = \sum_{i=0}^{6} p(i) \times precision(i)$$

where $p(i)$ is the portion/probability of samples in class i in the CV set.

You may compute overall precision using metrics.precision(). Note that metrics.accuracy will give you the ration of the total number of correct predictions for each class and the total number of samples.

What to submit?

You will submit a document that lists your source code and one screenshot of the overall precision value in the Spark Shell.

Show how you optimized Impurity, maxBins and other hyper-parameters to achieve the highest precision values in the testing dataset.