CART in Haskell

Genji Ohara

June 13, 2023

1 Preamble

```haskell
import Numeric.LinearAlgebra
import Prelude hiding ((<>))
import Text.ParserCombinators.Parsec
import Data.CSV
import Data.List

import DataSet
```

2 Data Type Definition

2.1 Data Space

Feature Space	$\mathcal{F} = \mathbb{R}^D$
Label Space	$\mathcal{L} = \{0, 1, \ldots, L-1\}$
Data Space	$\mathcal{D} = \mathcal{F} \times \mathcal{L}$

```haskell
type DataSet = [DataPoint]
```

2.2 Data Space

```
data Literal = Literal{lFeatureIdx :: Int, lValue :: Double} deriving Show

data Split = Split {sLiteral :: Literal, sScore :: Double} deriving Show

instance Eq Split where
    (Split l s) == (Split l’ s’) = s == s’

instance Ord Split where
    compare (Split l s) (Split l’ s’) = compare s s’

data Tree = Leaf {label :: Int} |
            Node {literal :: Literal, left :: Tree, right :: Tree}
            deriving Show
```

3 Gini Impurity

$$\mathrm{Gini} : \mathcal{L}^n \to \mathbb{R}$$

$$\mathrm{Gini}(L) = 1 - \sum_{i=0}^{L-1} p_i(L)^2$$

$$p_i(L) = \frac{1}{|L|} \sum_{l \in L} \mathbb{I}[l = i]$$

```
gini :: [Label] -> Double
gini labels = 1.0 - (sum $ map (^ 2) $ pList labels)

pList :: [Label] -> [Double]
pList labels = map (/ labelSetSize) $ map fromIntegral $ cntList labels 0
    where labelSetSize = fromIntegral $ length labels

cntList :: [Label] -> Int -> [Int]
cntList labels trg =
    if trg == labelNum
        then []
        else [length $ filter (== trg) labels] ++ (cntList labels $ trg + 1)
```

4 Search Best Split

4.1 Split Data

$$D_l(D, i, v) = \{(\boldsymbol{x}, y) \in D \mid x_i < v\}$$
$$D_r(D, i, v) = \{(\boldsymbol{x}, y) \in D \mid x_i \geq v\}$$

```
splitData :: DataSet -> Literal -> [DataSet]
splitData dataSet (Literal i v) = [lData, rData]
    where
        lData = [(DataPoint x y) | (DataPoint x y) <- dataSet, x !! i <= v]
        rData = [(DataPoint x y) | (DataPoint x y) <- dataSet, x !! i >  v]
```

4.2 Score Splitted Data

$$\mathrm{score}(D, i, v) = \frac{|D_l|}{|D|}\mathrm{gini}\left[D_l(D, i, v)\right] + \frac{|D_r|}{|D|}\mathrm{gini}\left[D_r(D, i, v)\right]$$

```
scoreLiteral :: DataSet -> Literal -> Split
scoreLiteral dataSet literal = Split literal score
    where
        score = sum $ map (weightedGini (length dataSet)) $ labelSet
        labelSet = map (map dLabel) $ splitData dataSet literal

weightedGini :: Int -> [Label] -> Double
weightedGini wholeSize labelSet = (gini labelSet) * dblDataSize / dblWholeSize
    where
        dblDataSize     = fromIntegral $ length labelSet
        dblWholeSize    = fromIntegral wholeSize
```

4.3 Search Best Split

$$\operatorname*{argmin}_{i,v} \mathrm{score}(D, i, v)$$

```
bestSplitAtFeature :: DataSet -> Int -> Split
bestSplitAtFeature dataSet i = foldr min (Split (Literal 0 0) 1) splitList
    where
        splitList   = [scoreLiteral dataSet l | l <- literalList]
        literalList = [Literal i (x !! i) | (DataPoint x y) <- dataSet]

bestSplit :: DataSet -> Split
bestSplit dataSet = foldr min (Split (Literal 0 0) 1) splitList
    where splitList = [bestSplitAtFeature dataSet f | f <- [0,1..featureNum-1]]
```

```
growTree :: DataSet -> Int -> Int -> Tree
growTree dataSet depth maxDepth =
    if depth == maxDepth
        then Leaf $ majorLabel dataSet
        else Node literal leftTree rightTree
    where
        literal         = sLiteral $ bestSplit dataSet
        leftTree        = growTree lData depth + 1 maxDepth
        rightTree       = growTree rData depth + 1 maxDepth
        [lData, rData]  = splitData dataSet literal
```

5 Main

```
main = do
    rawDataSet <- parseFromFile csvFile "../data/iris/iris.data"
    let dataSet = either (\x -> []) processData rawDataSet
    print $ bestSplit dataSet
    print $ scoreLiteral dataSet $ Literal 2 2.45
```